Project team #25 - Airport Database Management System

Team Members:
Akash Khiste
Shubham Gupta
Stephen Dean
Gokul Rajakumar

Objective:-

The objective of this project is to design and implement airport database management system which is accessible by airport personnel and airport employees to monitor day to day operations and carter the informative needs of passengers. The collection and storage of airport information into the database ensures maximum operational productivity and provides platform for managing the data securely. The purpose of our database is to regulate air traffic within an airport meticulously.

Scope:-

Develop a database management system that is efficient in carrying out certain operations which includes every minute detail of the airport. Research about what all things needs to be specified within the database with pragmatic approach that is feasible for employees. This includes details of staff, employees, parking facilities, Shops etc. Meeting with the stakeholders along with airport authorities to finalize the requirements for new database. Eventually, it results in the creation of database management system.

Content:-

This project will include significant data relevant for airport operations such as :

- Flight arrivals/departures
- Airline Information
- Airport employee details (*Help desk, ground staff, security staff, etc.*)
- Facilities of Airport (E.g. Private lounge, Waiting Rooms, Shops)
- Transportation within the terminals
- Paid vehicle parking system

PROJECT ENVIRONMENT

We will be using Amazon Relational Database Service and Database Instance class db.t2.micro for creating all the database tables. We will be using MySQL Workbench 8.0

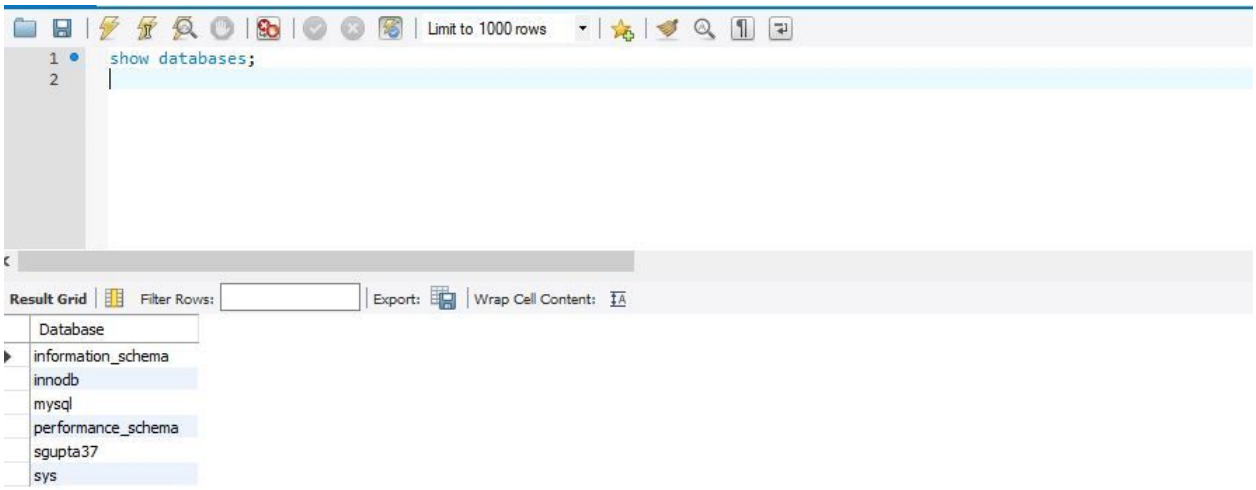
Demonstration of Database access through UI :

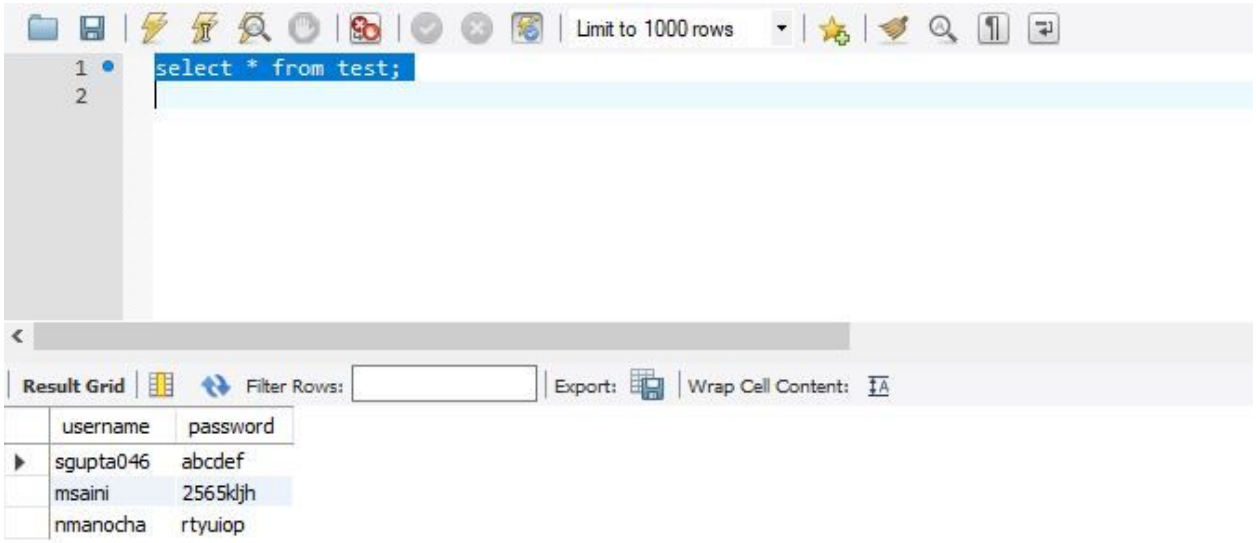
We used Java Swing components mainly JFrame to develop a minimal UI to access our AW S Database. This demonstration includes a test table in database 'sgupta37' which stores Username & password and we will be accessing through UI to validate the credentials and display successful login or exception.

Code Snippet:

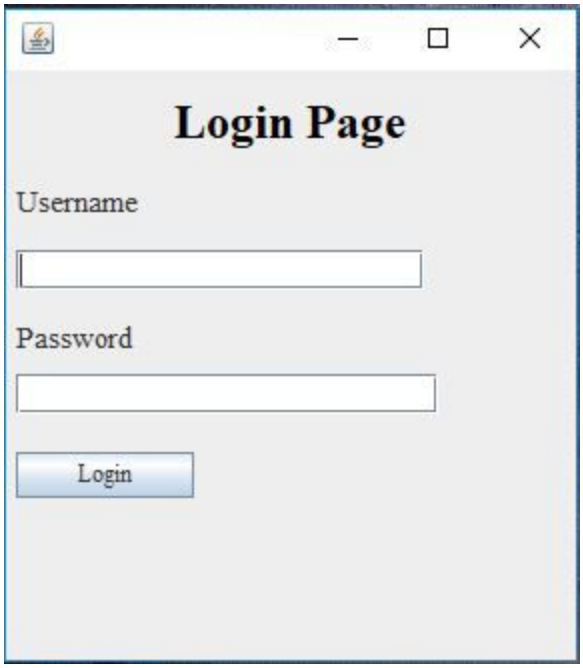
```
JButton btnNewButton = new JButton("Login");
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            Connection con=DriverManager.getConnection("jdbc:mysql://fall2018dbshubham.cutrzcvjwnrd.us-east-2.rds.amazonaws.com:3306/sgupta37",
            Statement stmt=con.createStatement();
            String sql="Select * from test where username='"+user.getText()+"' and password='"+pass.getText().toString()+"'";
            ResultSet rs=stmt.executeQuery(sql);
            if (rs.next())
                JOptionPane.showMessageDialog(null, "Login Successfully..");
            else
                JOptionPane.showMessageDialog(null, "Incorrect username or password");
            con.close();
        }catch (Exception e) {System.out.println(e);}
    }
});
```

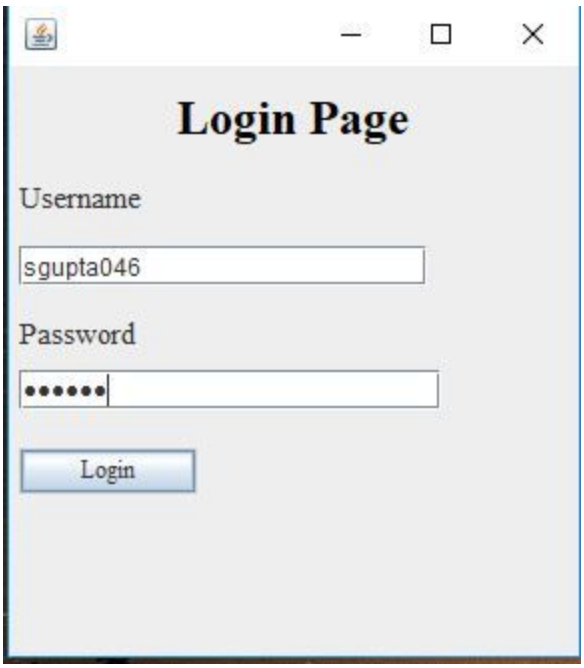
Database Snippet:

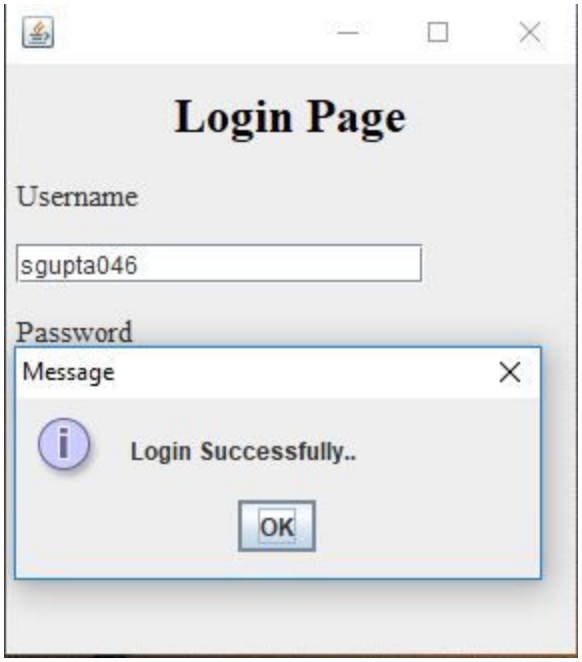



Limit to 1000 rows
1
2
select * from test;
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
username
password
sgupta046
abcdef
msaini
2565kljh
nmanocha
rtyuiop

UI:


Login Page
Username
Password
Login


Login Page
Username
sgupta046
Password
Login



HIGH LEVEL REQUIREMENTS

Initial user roles

User Role	**Description**
passengers	Passengers are allowed the most basic access in checking the time and status of their flight
airport_administrator_support (AAS)	Members of the Airport Administrator Support are given access to a complete schedule of arrivals and departures (Help Desk)
Ramp Planner	Members of airlines are allowed to update the flight information and schedules in airport database.
air_traffic_control (ATC)	Members of ATC are capable of canceling and delaying scheduled flights
airport_manager	Airport Manager can manage the Airport Employee Details
facilities_manager	Facilities Manager manage the Airport Facilities such as Restaurants, Lounge, Shops, Parking etc.
ground_staff	Ground Staff can manage the transportation details within the terminals

database_admin(DBA)	Database Admin who will have complete access for Airport Database Management

Initial user story descriptions

Story ID	Story description
US1	As a passenger, I want to see my flight status so that I know if it is delayed or on time.
US2	As a member of the Airport Administrator Support, I want to see schedule of all the flights listed for today so I can inform passengers.
US3	As a passenger, I want to browse features of airport such as Shops, Lounge, Parking, etc.
US4	As a ramp planner, I want to update flight information.
US5	As an ATC, I want to update flight information for any changes
US6	As a ground_staff, I want to update the transportation details within the terminals.
US7	As an airport_manager, I want to add/update airport employee details.
US8	As a facilities_manager, I want to add/update shops & restaurants, lounge.
US9	As a facilities_manager, I want to keep a track of available parking slots.
US10	As an Airport Administrator Support, I want to log in and out of my account with my credentials.

HIGH LEVEL CONCEPTUAL DESIGN

Entities:

1. Passenger
2. Flight_Details
3. Employee*

4. Login_Account
5. Parking
6. Transportation_Details
7. Terminal_Details
8. Shops_Restaurants
9. Lounge

Note: The entity Employee* are specialised into airport_manager, facilities_manager, airport_admin_suport, ATC, and ground_staff and included each of them.

Relationships:

Passenger views Flight_Details
Airport Administrator Support browses Flight_Details
Employee updates Flight_Detail
Passenger browses Shops_Restaurants
Passenger browses Lounge
Passenger browses Parking
Employee manages Employee
Employee manages Shops_Restaurants details
Employee manages Lounge details
Employee manages Parking details
Employee update Transportation_Details
Airport Administrator Support logs in/log out of Login_Account

Sprint 1

REQUIREMENTS

Highlighted user stories are considered for the current sprint.

Story ID	**Story description**

US1	As a passenger, I want to see my flight status so that I know if it is delayed or on time.
US2	As a ramp planner, I want to update scheduled flight information.
US3	As a member of the Airport Administrator Support, I want to see schedule of all the flights listed for today so I can inform passengers.
US4	As an Airport Administrator Support, I want to log in and out of my account with my credentials.
US5	As ATC I want to update flight change information
US6	As a passenger, I want to browse the features of the airport such as Shops, Lounge, Parking, etc.
US7	As a ground_staff, I want to update the transportation details within the terminals.
US8	As an airport_manager, I want to add/update airport employee details.
US9	As a facilities_manager, I want to add/update shops & restaurants, lounge.
US10	As a facilities_manager, I want to keep a track of available parking slots.

CONCEPTUAL DESIGN

Entity: **Passenger**
Attributes:
- passenger_id
- name [composite]
  - first_name
  - last_name
- pnr_no
- ticket _no
- ticket_class
- passenger_type
- seat_no
- contact [composite]

email
phone
meal_type

Note :-
ticket_class represents class of flight ticket as economy and business class.
passenger_type refers to type of passengers as adult, child, passenger with medical condition.

Entity: **Flight**
Attributes:
flight_id
airline_no
airline_name
source
destination
date
departure_time
arrival_time
gate_no
terminal_no
status

Note: There can be many flights with the same airline_no. So in order to uniquely identify each flight we kept flight_id as the primary key.

Entity: **Employee**
Attributes:
employee_id
designation
emp_name [composite]
emp_firstname
emp_lastname
address [composite]
street_address
city
state
country
zipcode
contact [composite]
email
phone
salary

Entity: **Account**
Attributes:
account_id
username
Password

Relationship: **Passenger** accesses **Flight**
Cardinality: One to Many
Participation:
Passenger has partial participation
Flight has partial participation

Relationship: **Employee** updates **Flight**
Cardinality: Many to Many
Participation:
Employee has partial participation
Flight has total participation

Note: Both the Ramp Planner and the ATC can update the flight details of the same flight.

Relationship: **Employee** logs into **Account**
Cardinality: One to One
Participation:
Employee has total participation
Account has total participation

LOGICAL DESIGN

Table: **Flight**
Columns:
flight_id
airline_no
airline_name
source
destination
date

departure_time
arrival_time
gate_no

Table: **Passenger**
Columns:
passenger_id
first_name
last_name
pnr_no
ticket_no
ticket_class
passenger_type
seat_no
email
phone_no
Meal_type
flight_id [foreign key references flight_id of Flight table]

Justification: ticket_class refers to Economy/Business
passenger_type refers to adult or child
flight_id has been added as a foreign key to relate the flight details of a passenger

Table: **Employee**
Columns:
employee_id
designation
emp_firstname
emp_lastname
street_address
city
state
country
zip_code
email
phone
salary

Justification: Employee is a generalized table for all employees of the airport which includes an attribute designation which is basically a specialisation of different types

of employees. For example - Ramp Planner, ATC and Airport Administrator Suupport (Help Desk), etc.

Table: **Account**

Columns:

account_id

username

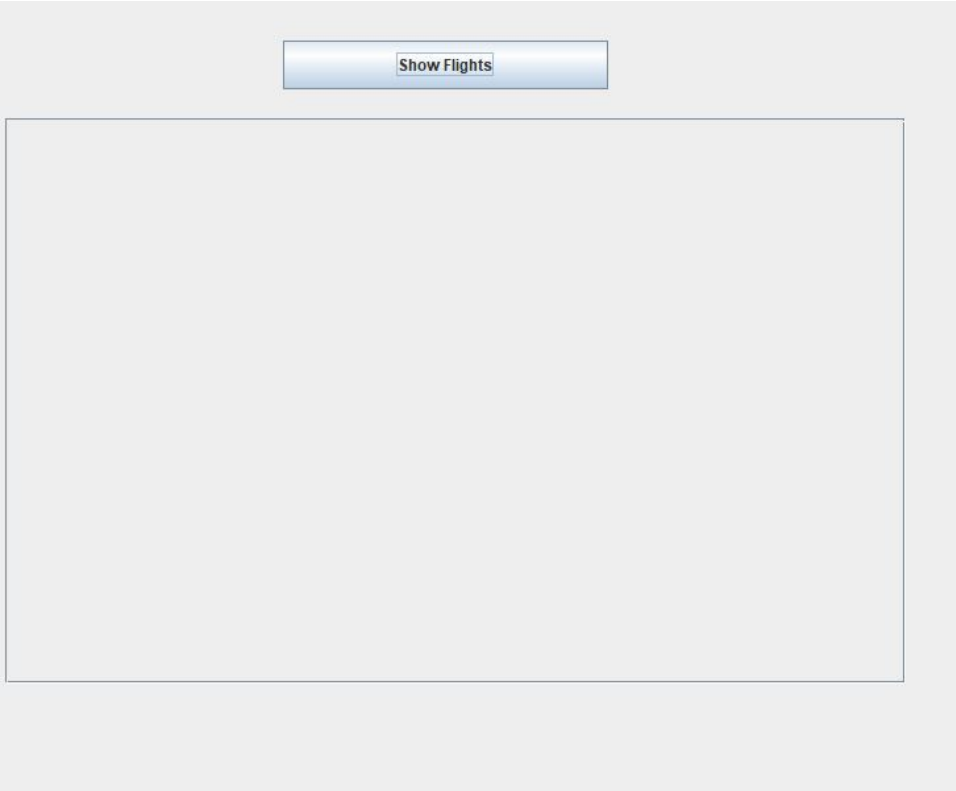
password

emp_id [foreign key; references employee_id of Employee Table]

SQL QUERIES

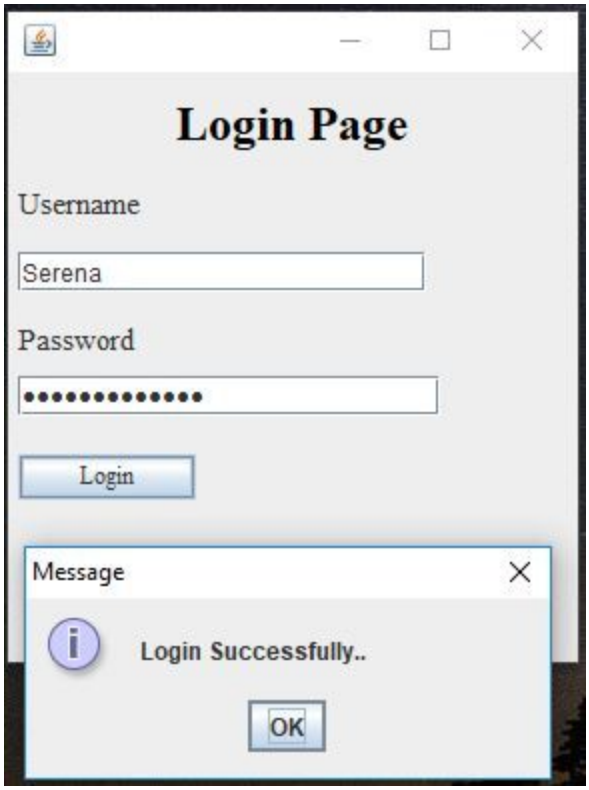
Demonstration of selected user queries for this sprint :

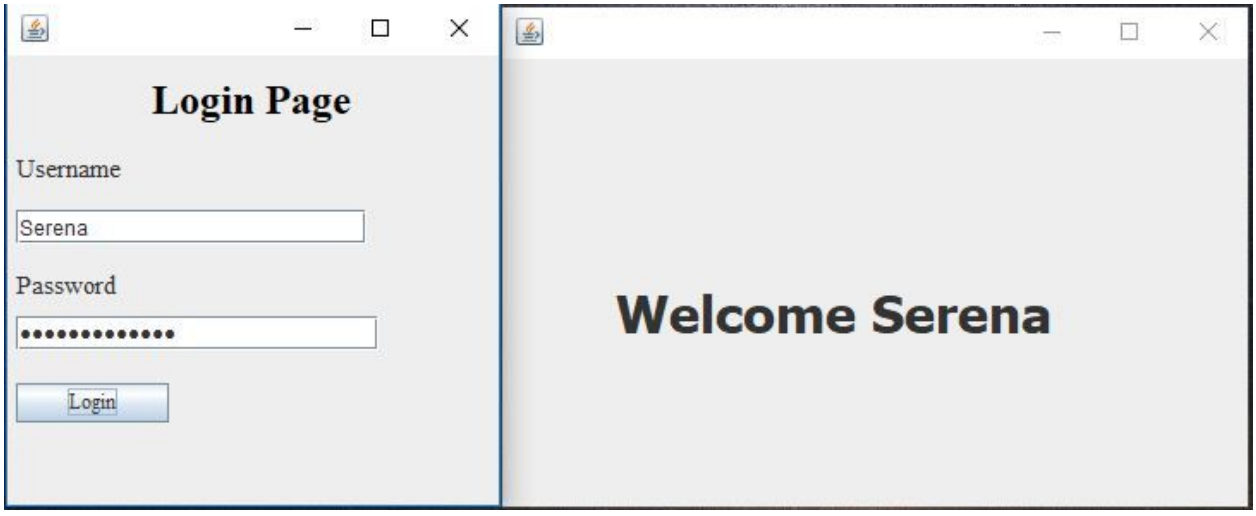
1) List of flights browsed by Passenger or Help Desk (AAS) :



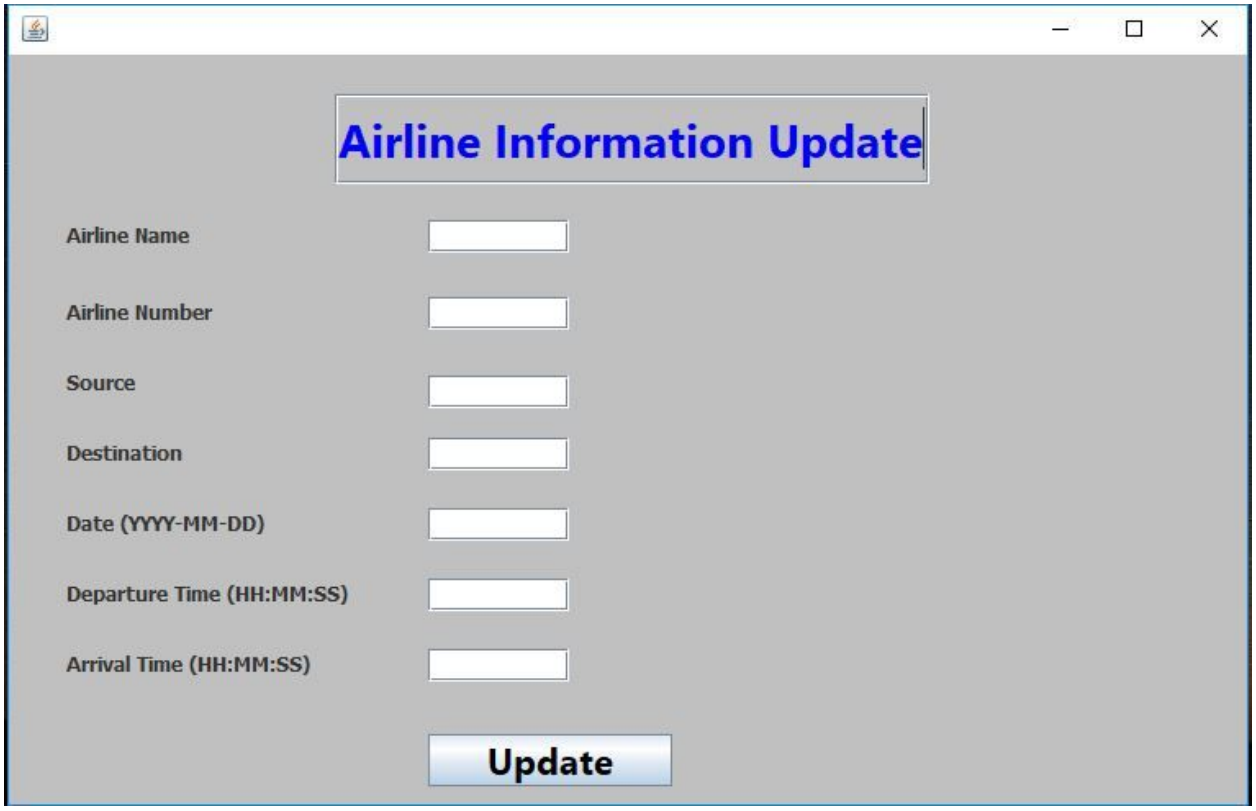
flight_id	airline_no	airline_na...	source	destination	date	departure...	arrival_time	gate_no	Terminal_...
1	JB-123	Jet Blue	San Anton...	San Anton...	2018-09-24	13:52:58	02:14:44	M3	3
2	DA-543	Delta Air li...	Charlotte, ...	New York ...	2018-10-04	21:15:26	04:35:07	F5	9
3	AA-334	American ...	Seattle, WA	Charlotte, ...	2019-08-05	12:21:43	16:11:36	T8	7
4	JB-987	Jet Blue	Charlotte, ...	Charlotte, ...	2018-04-12	22:22:42	12:24:03	J4	4
5	JB-65	Jet Blue	Dallas, TX	New York ...	2018-01-15	07:24:21	00:24:49	W3	4
6	JB-77	Jet Blue	San Anton...	Boston, MA	2019-07-27	14:08:25	22:24:41	A9	1
7	JB-387	Jet Blue	Memphis,...	Memphis,...	2019-03-09	13:32:13	04:35:01	B4	1
8	AA-776	American ...	Philadelp...	Charlotte, ...	2018-02-06	05:31:04	06:27:41	V3	7
9	SA-678	Southwes...	Philadelp...	Chicago, IL	2018-09-17	02:24:41	01:33:02	O3	3
10	SA-621	Southwes...	Charlotte, ...	Boston, MA	2019-09-26	20:56:39	19:27:45	D2	3
11	AA-76	American ...	Chicago, IL	Philadelp...	2018-01-27	07:31:51	09:04:16	W1	2
12	DA-53	Delta Air li...	Dallas, TX	Chicago, IL	2019-01-30	15:53:00	05:18:50	H4	4
13	SA-21	Southwes...	Philadelp...	Chicago, IL	2019-07-24	17:27:01	03:30:49	O4	8
14	AA-276	American ...	Dallas, TX	Memphis,...	2019-02-08	16:45:22	03:54:49	Y1	4
15	DA-543	Delta Air li...	Memphis,...	Boston, MA	2019-04-03	06:48:12	05:12:01	E2	6
16	AA-676	American ...	New York ...	Philadelp...	2019-08-26	12:31:09	02:51:53	J0	5
17	JB-327	Jet Blue	New York ...	Seattle, WA	2018-10-30	00:04:55	12:38:59	X6	1
18	JB-887	Jet Blue	Memphis,...	Charlotte, ...	2018-09-16	10:30:48	23:47:55	R2	5
19	AA-996	American ...	Charlotte, ...	Seattle, WA	2018-01-05	07:19:25	19:19:27	X5	9
20	SA-241	Southwes...	Charlotte, ...	San Anton...	2018-10-22	04:23:54	00:03:23	R3	1
21	AA-596	American ...	Charlotte, ...	Boston, MA	2018-01-15	05:25:20	07:30:20		

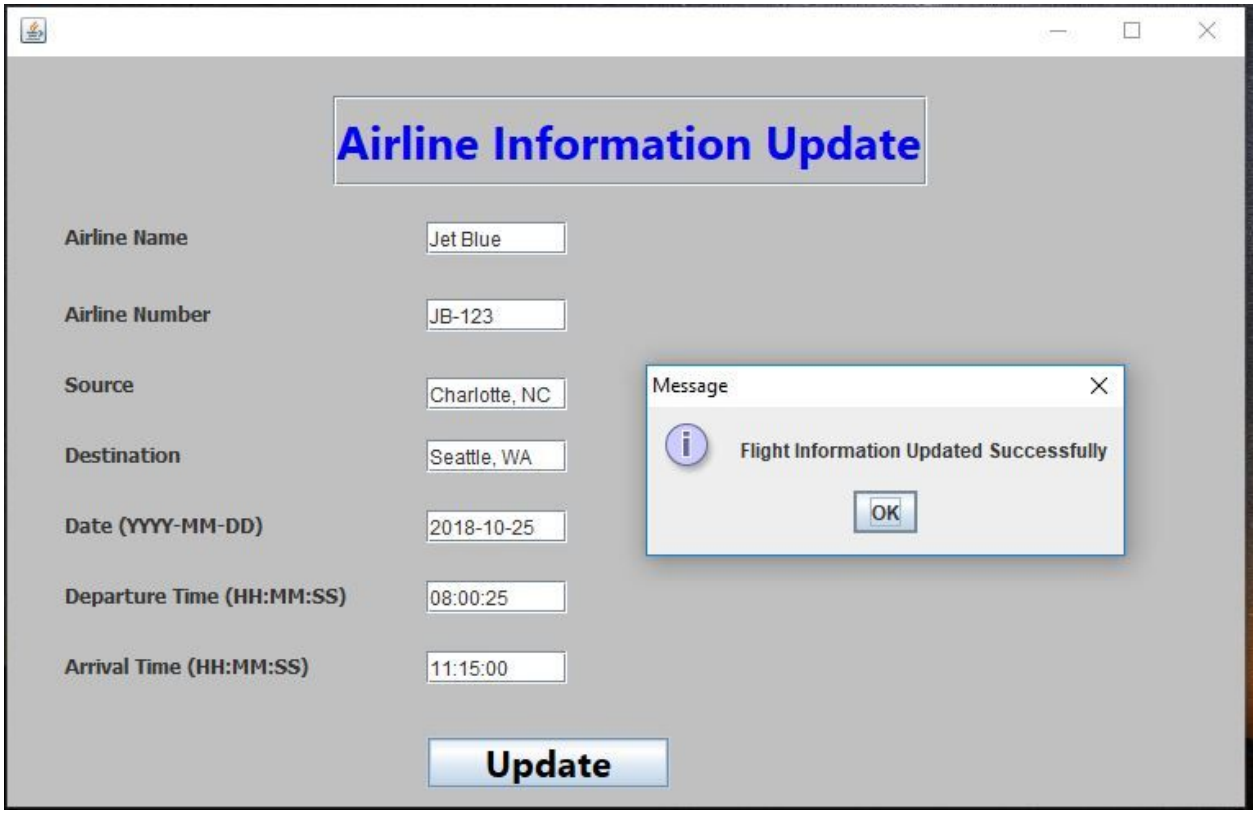
2) Employee (AAS - Help Desk) logging into their account :





3) Flight Details updated by Ramp Planner :


Airline Information Update
Airline Name
Airline Number
Source
Destination
Date (YYYY-MM-DD)
Departure Time (HH:MM:SS)
Arrival Time (HH:MM:SS)
Update



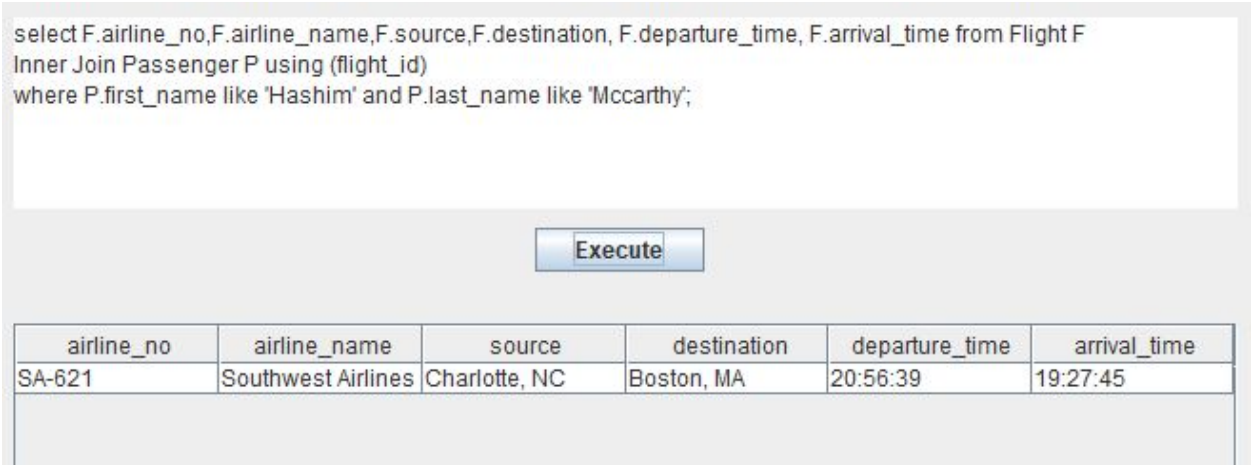
Moreover, we can run any query using the below text box.

Sample Queries :

1) A passenger named ‘Hashim Mccarthy’ wants to view details of his flights :

   Query :

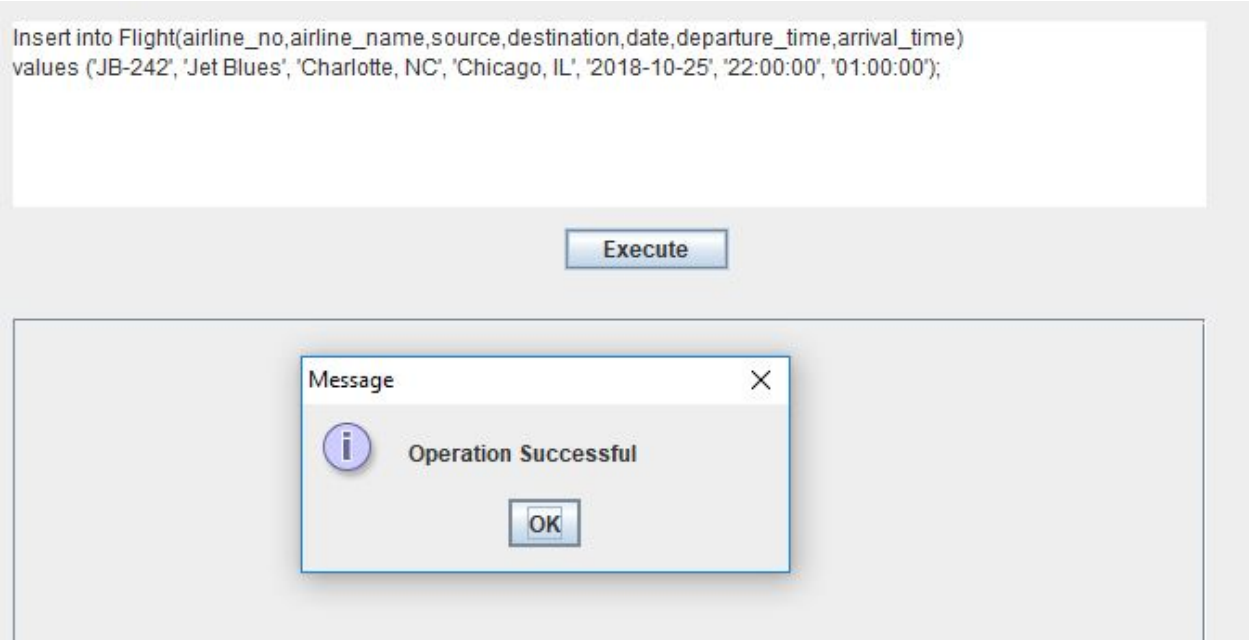
   select F.airline_no,F.airline_name,F.source,F.destination, F.departure_time, F.arrival_time from Flight F
   Inner Join Passenger P using (flight_id)
   where P.first_name like 'Hashim' and P.last_name like 'Mccarthy';

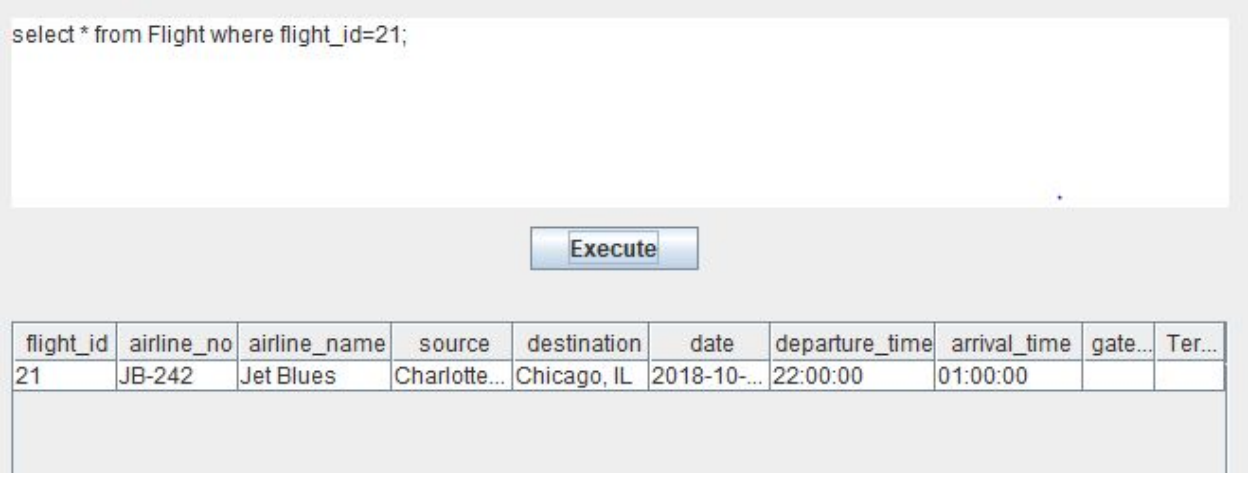


2) A Ramp Planner wants to insert details of a Flight (He can only update basic flight details, not the terminal and gate details)

   Query:

   Insert into
   Flight(airline_no,airline_name,source,destination,date,departure_time,arrival_time)
   values ('JB-242', 'Jet Blues', 'Charlotte, NC', 'Chicago, IL', '2018-10-25', '22:00:00', '01:00:00');

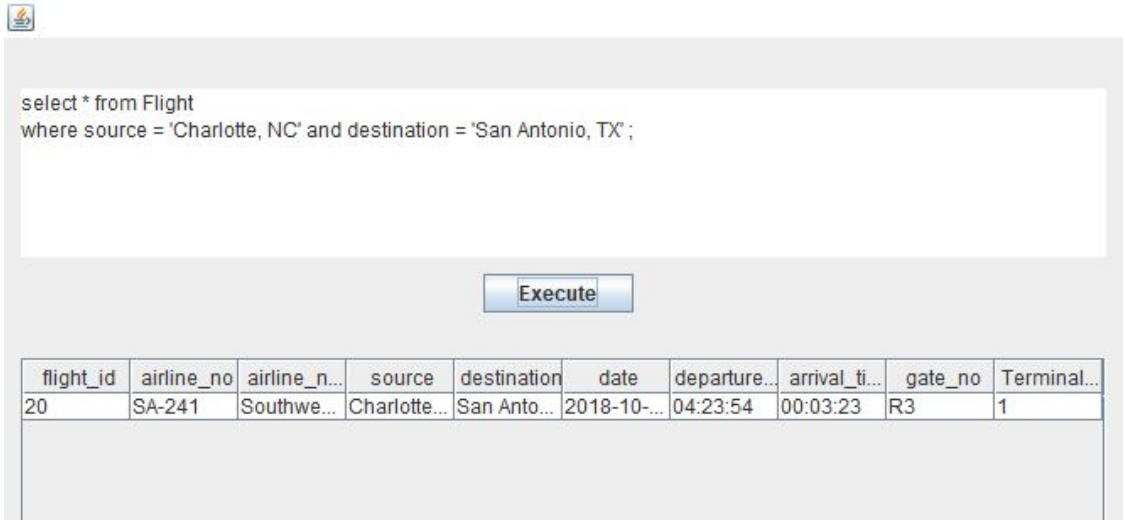




3) An Airport Administrator Support Employee wants to inform a passenger about a flight departing from Charlotte to Chicago.

Query :

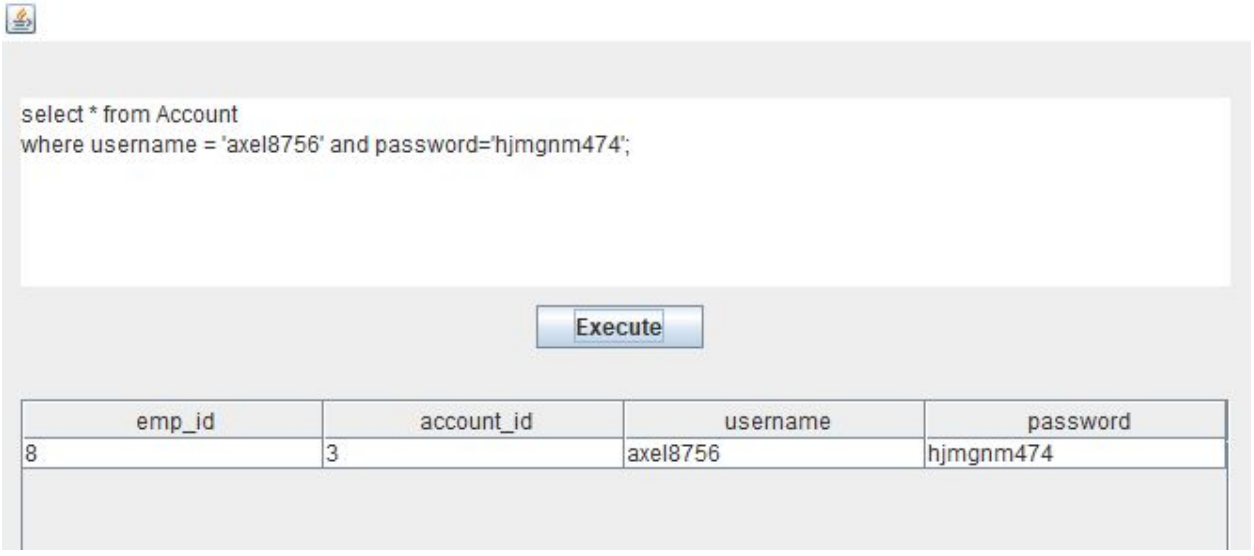
```
select * from Flight
where source = 'Charlotte, NC' and destination = 'San Antonio, TX' ;
```



4) An Airport Administrator Support wants to log in to their account.

Query :
select * from Account
where username = 'axel8756' and password='hjmgnm474';

If this query is executed successfully i.e. username and password of an employee matches in the database then they are allowed to log in, else not.



Project: Sprint 2 - Database design and implementation

Story ID	Story description
US1 (Done)	As a passenger, I want to see my flight status so that I know if it is delayed or on time.
US2 (Done)	As a ramp planner, I want to update scheduled flight information.
US3 (Done)	As a member of the Airport Administrator Support, I want to see schedule of all the flights listed for today so I can inform passengers.
US4 (Done)	As an Airport Administrator Support, I want to log in and out of my account with my credentials.
US5	As ATC I want to update flight change information
US6	As a passenger, I want to browse the features of the airport such as Shops, Lounge, Parking, etc.
US10	As a facilities_manager, I want to keep a track of available parking slots.
US9	As a facilities_manager, I want to add/update shops & restaurants, lounge.
US7	As a ground_staff, I want to update the transportation details within the terminals.
US8	As an airport_manager, I want to add/update airport employee details.

Stories marked in green were completed in Sprint 1 and highlighted in yellow are considered for the current sprint.
As per the feedback received in Sprint 1, we have changed the relationship between entities Flight & Passenger [Highlighted]

CONCEPTUAL DESIGN

Entity: **Passenger**
Attributes:

- passenger_id
- name [composite]
  - first_name
  - last_name
- pnr_no
- ticket _no
- ticket_class
- passenger_type
- seat_no
- contact [composite]
  - email
  - phone
- meal_type

Note :-
ticket_class represents class of flight ticket as economy and business class.
passenger_type refers to type of passengers as adult, child, passenger with medical condition.

Entity: **Flight**
Attributes:

- flight_id
- airline_no
- airline_name
- source
- destination
- date
- departure_time
- arrival_time
- gate_no
- terminal_no
- status

Note: There can be many flights with the same airline_no. So in order to uniquely identify each flight we kept flight_id as the primary key.

Entity: **Employee**
Attributes:

- employee_id
- designation
- emp_name [composite]

emp_firstname
emp_lastname
address [composite]
street_address
city
state
country
zipcode
contact [composite]
email
phone
salary

Entity: **Account**
Attributes:
account_id
username
Password

Entity: **Shops_Restaurants**
Attributes:
shop_id
name
terminal_no
area
category

Note: Area- before security or after security….. etc

Entity: **Parking**
Attributes:
lot_id
terminal_no
capacity
availability

Entity: **Lounge**
Attributes:
lounge_id

type
terminal_no
capacity
availability

Note: type refers to the type of lounge (regular or premium).

Relationship: **Passenger** accesses **Flight**
Cardinality: Many to Many
Participation:
Passenger has partial participation
Flight has partial participation

Relationship: **Employee** updates **Flight**
Cardinality: Many to Many
Participation:
Employee has partial participation
Flight has total participation

Note: Both the Ramp Planner and the ATC can update the flight details of the same flight.

Relationship: **Employee** logs into **Account**
Cardinality: One to One
Participation:
Employee has total participation
Account has total participation

Relationship: **Employee** updates **Parking**
Cardinality: Many to Many
Participation:
Passenger has partial participation
Parking has partial participation

Note: Here Employee refers to Facilities Manager

Relationship: **Employee** updates **Shops_Restaurants**
Cardinality: Many to Many
Participation:
Employee has partial participation

Shops_Restaurants has partial participation

Note: Here Employee refers to Facilities Manager

Relationship: **Employee** updates **Lounge**
Cardinality: Many to Many
Participation:
Passenger has partial participation
Lounge has partial participation

Note: Here Employee refers to Facilities Manager

LOGICAL DESIGN WITH NORMAL FORM IDENTIFICATION

Table: **Flight**
Columns:
flight_id
airline_no
airline_name
source
destination
Highest normalization level: 4NF

Note :- We had Flight Table in Sprint 1 that had all the attributes. While normalisation, we separated it into two tables : Flight & FlightDetails.

Table : **FlightDetails**
Columns:
flight_id [foreign key:references Flight table]
date
departure_time
arrival_time

gate_no
Terminal_no

Justification for PK : flight_id+date can uniquely identify any row of this table. If flight_id was primary key then we can not have multiple entries for same flight even if that flight is scheduled on daily basis.

Note: FlightDetails was separated from Flight so that ATC/Ramp planner can update departure_time and arrival_time and reduce data redundancy.

Highest normalization level: 4NF

Table: **Passenger**

Columns:

passenger_id
first_name
last_name
passenger_type
email
phone_no

~~flight_id [foreign key references flight_id of Flight table]~~

Justification: passenger_type refers to adult or child

Highest normalization level: 4NF

Justification: Passenger table is separated into two tables in process of normalization resulting tables are passenger and passengesflight.

Table : **PassengerFlight**

Columns:

flight_id [foreign key references flight_id of Flight table]
passenger_id [foreign key references passenger_id of Passenger table]
pnr_no
ticket_no
ticket_class
seat_no
meal_type

Primary key : pnr_no+flight_id+passenger_id

Assuming that the PNR number for connecting flights might be same , we chose combination of these keys as a PK.

Justification: ticket_class refers to Economy/Business
Highest normalization level:4NF

Table: **Employee**
Columns:
employee_id
designation
emp_firstname
emp_lastname
street_address
city
state
country
zip_code
email
phone
salary

Note: Employee is a generalized table for all employees of the airport which includes an attribute designation which is basically a specialization of different types of employees. For example - Ramp Planner, ATC and Airport Administrator Support (Help Desk), etc.

Highest normalization level: 2NF
Justification : In Employee table columns street_address, city, state, country are dependent on zip_code , however considering the requirement of this specific database keeping all columns in one table is more efficient.

Table: **Account**
Columns:
account_id
username
password
emp_id [foreign key; references employee_id of Employee Table]

Highest normalization level: 4NF

Table: **ShopRestCat**
Columns:
shop_id
name

category

Highest normalization level: 4NF
Justification : ShopRestCat was added to remove the dependency of name and category and to reduce data redundancy

Table: **ShopsRestaurants**
Columns:
shop_id foreign key references ShopRestCat(shop_id)
terminal_no
area

Highest normalization level: 4NF
Justification : Table ShopsRestaurants was separated into two tables in order to remove the dependency resulting table ShopsRestaurants is in 4nf.

Table: **LoungeCat**
Columns:
lounge_id
type
Highest normalization level: 4NF
Justification : LoungeCat was separated from Lounge to reduce data redundancy and address the dependency of lounge_id and lounge type

Table:**Lounge**
Columns:
lounge_terminal_id
lounge_id (foreign key, references lounge_id of LoungeCat)
terminal_no
capacity
availability

Primary Key : **lounge_terminal_id+lounge_id+terminal_no**

Combination of these 3 keys has been chosen as primary key to put a constraint that only entry of a lounge type is allowed in a terminal.

Highest normalization level: 4NF
Justification : Table Lounge was separated into two tables in order to eliminate dependency, resulting table Lounge in 4nf.

Table: **Parking**

Columns:

lot_id
terminal_no
capacity
availability

Highest normalization level: 4NF

Note: We had many to many relationships for this sprint, which translates to each of the below separate tables in our Relational Model.

Table: **EmpFlight**

Columns:

flight_id [foreign key references lot_id of Flight table]
employee_id [foreign key references employee_id of Employee table]

Highest normalization level: 4NF

Table: **EmpParking**

Columns:

lot_id [foreign key references lot_id of Parking table]
employee_id [foreign key references employee_id of Employee table]

Highest normalization level: 4NF

Table: **EmpShopRest**

Columns:

shop_id [foreign key references shop_id of ShopRestCat table]
employee_id [foreign key references employee_id of Employee table]

Highest normalization level: 4NF

Table: **EmpLounge**

Columns:

lounge_terminal_id [foreign key references lounge_terminal_id of Lounge table]
employee_id [foreign key references employee_id of Employee table]

Highest normalization level: 4NF

SQL QUERIES

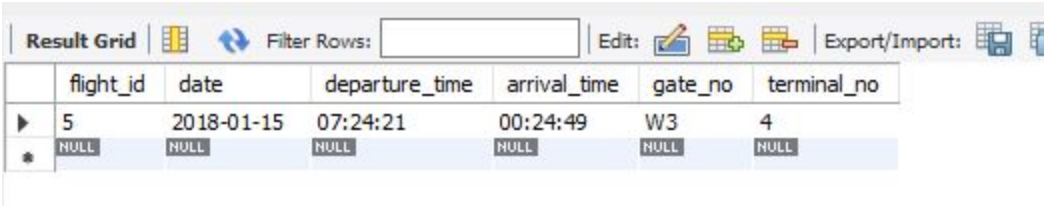
Demonstration of selected user queries for this sprint :

1) **Jet Blue Flight no. "JB-65" coming from "Dallas, TX" was supposed to land at 00:24:00 but now is delayed by 36 mins and ATC Employee needs to update this info.**

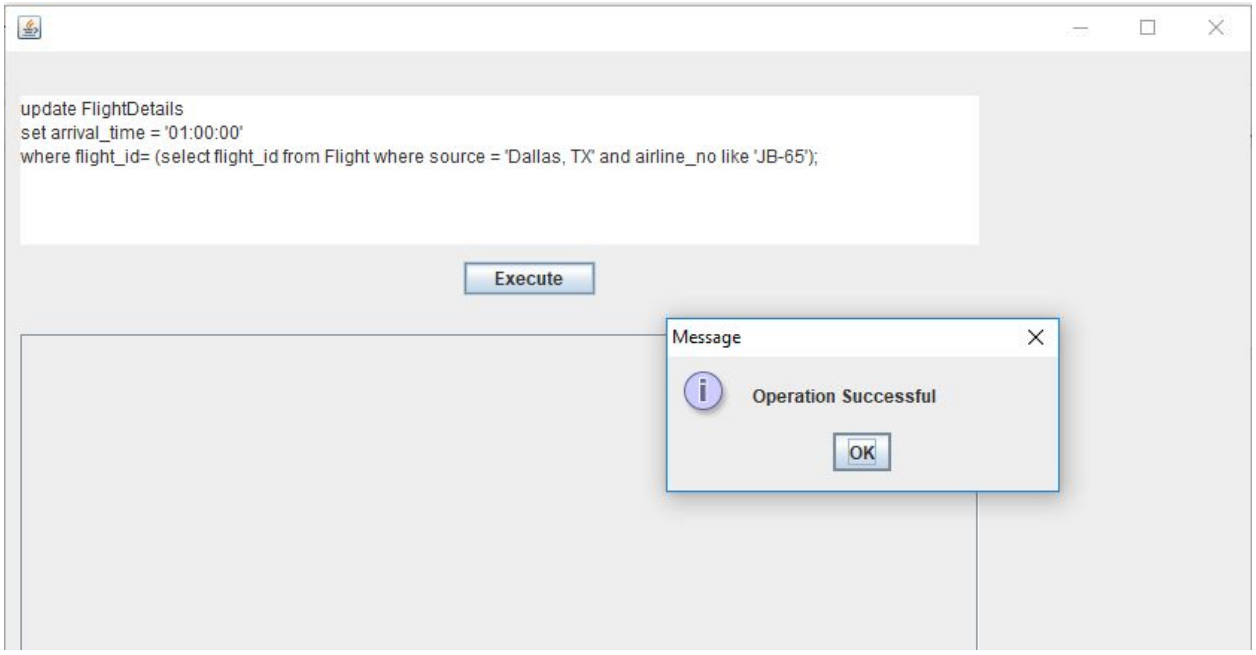
Query :

```
update FlightDetails
set arrival_time = '01:00:00'
where flight_id= (select flight_id from Flight where source = 'Dallas, TX' and airline_no like 'JB-65');
```

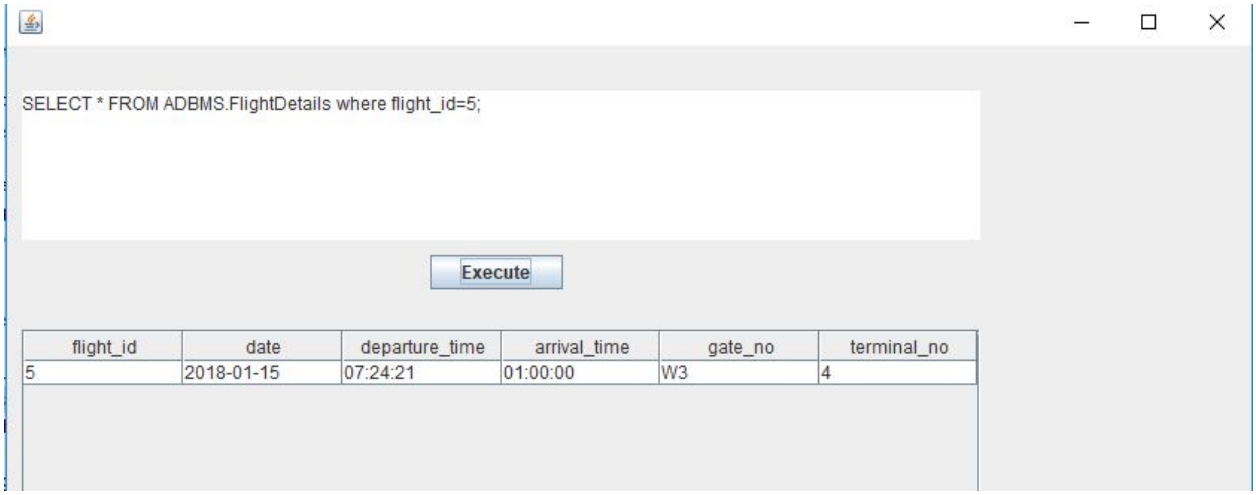
Initially:



flight_id	date	departure_time	arrival_time	gate_no	terminal_no
5	2018-01-15	07:24:21	00:24:49	W3	4
NULL	NULL	NULL	NULL	NULL	NULL



After update:

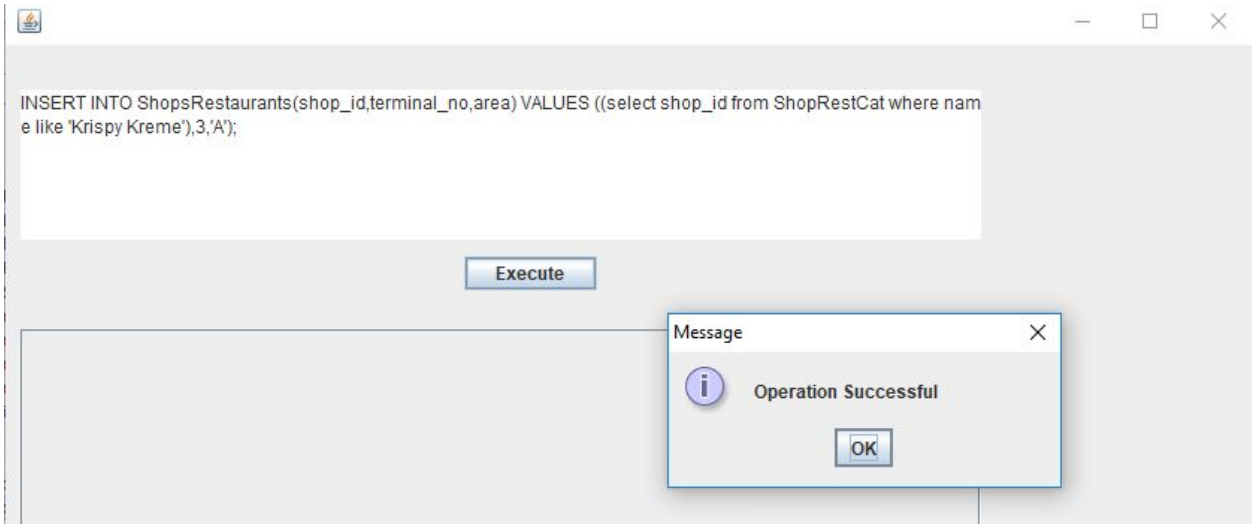


2) **A facilities manager wants to check parking slots availability in terminal 10.**

Query:
select availability from Parking where terminal_no=10;

3) **A facilities manager wants to add a shop "Krispy Kreme" at Terminal 6 to the airport database**

Query:
INSERT INTO ShopsRestaurants(shop_id,Terminal_no,area) VALUES ((select shop_id from ShopRestCat where name like 'Krispy Kreme'),3,'A');



4) **A facilities manager wants to update the existing shop at a terminal in the airport database**

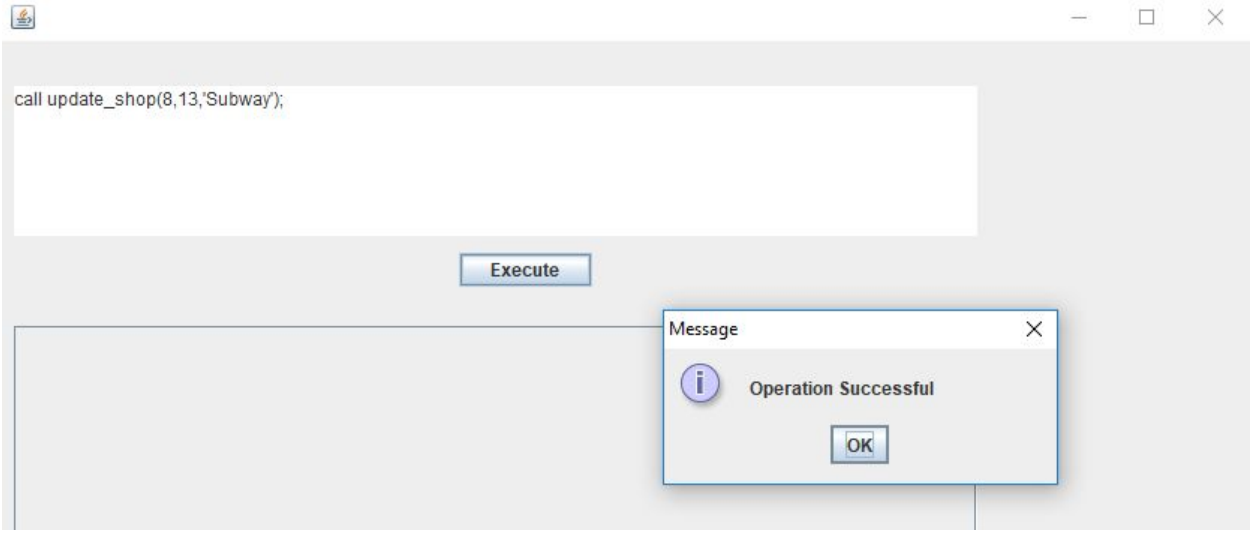
We created a stored procedure for this and called the procedure from the UI and verified the output.

Procedure definition :

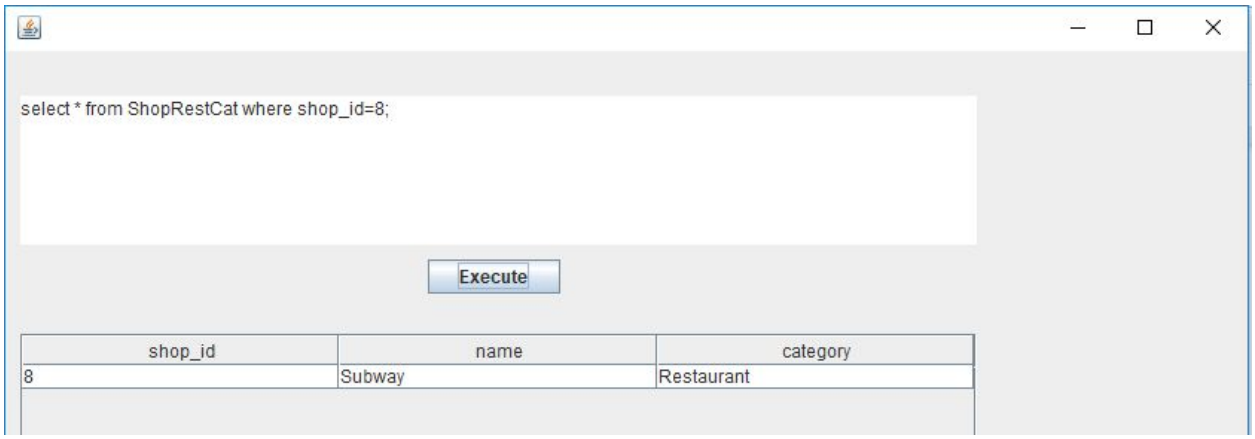
```
DELIMITER $$
CREATE PROCEDURE update_shop(IN shopID int , IN empID int, IN shopName varchar(255))
BEGIN
UPDATE ShopsRestCat SET name = shopName WHERE shop_id = shopID;
Insert into EmpShopRest values (shopID,empID);
END$$
DELIMITER ;
```

Calling procedure:

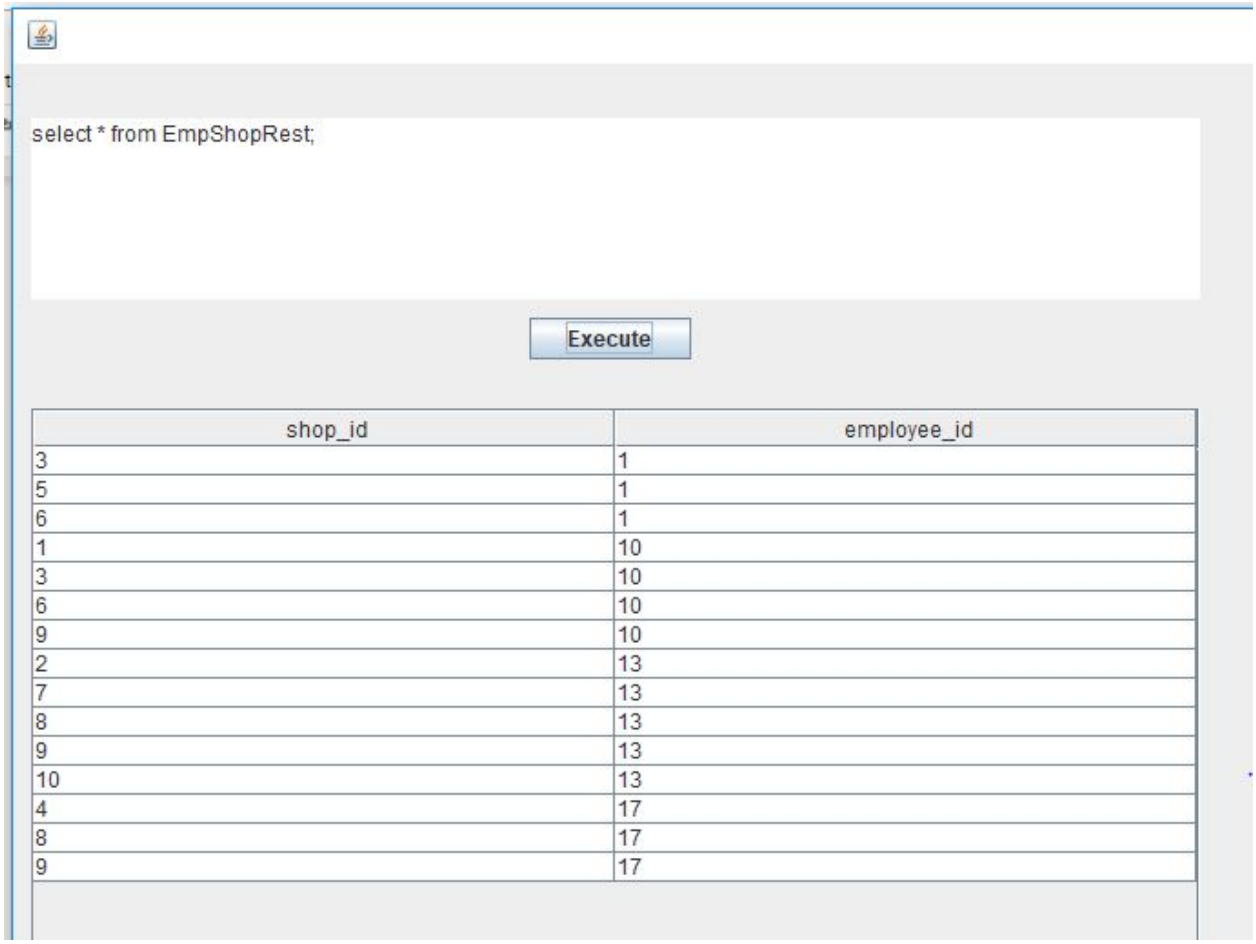
call update_shop(8,13,'Subway');



select * from ShopsRestaurants where shop_id=8;



select * from EmpShopRest;



Project: Sprint 3 - Database design and implementation

Story ID	**Story description**
US1 (Done)	As a passenger, I want to see my flight status so that I know if it is delayed or on time.
US2	As a ramp planner, I want to add scheduled flight information.

(Done)	
US3 (Done)	As a member of the Airport Administrator Support, I want to see schedule of all the flights listed for today so I can inform passengers.
US4 (Done)	As an Airport Administrator Support, I want to log in and out of my account with my credentials.
US5 (Done)	As ATC I want to update flight change information
US6 (Done)	As a passenger, I want to browse the features of the airport such as Shops, Lounge, Parking, etc.
US7 (Done)	As a facilities_manager, I want to keep a track of available parking slots.
US8 (Done)	As a facilities_manager, I want to add/update shops & restaurants, lounge, parking.
US9	As a ground_staff, I want to update the transportation details within the terminals.
US10	As an airport_manager, I want to add/update airport employee details.

Stories marked in green were completed in Sprint 1 and orange were done in Sprint 2. US9 marked in yellow is part of this sprint. We chose only one user story for this sprint to focus more on the index and the stored programs.

CONCEPTUAL DESIGN

Entity: **Passenger**
Attributes:
- passenger_id
- name [composite]
  - first_name
  - last_name
- pnr_no
- ticket _no
- ticket_class

passenger_type
seat_no
contact [composite]
    email
    phone
meal_type

Note :-
ticket_class represents class of flight ticket as economy and business class.
passenger_type refers to type of passengers as adult, child, passenger with medical condition.

Entity: **Flight**
Attributes:
<u>flight_id</u>
airline_no
airline_name
source
destination
date
departure_time
arrival_time
gate_no
terminal_no
status

Note: There can be many flights with the same airline_no. So in order to uniquely identify each flight we kept flight_id as the primary key.

Entity: **Employee**
Attributes:
<u>employee_id</u>
designation
emp_name [composite]
    emp_firstname
    emp_lastname
address [composite]
    street_address
    city
    state
    country
    zipcode
contact [composite]

email
phone
salary

Entity: **Account**
Attributes:
account_id
username
Password

Entity: **Shops_Restaurants**
Attributes:
shop_id
name
terminal_no
area
category

Note: Area- before security or after security….. etc

Entity: **Parking**
Attributes:
lot_id
terminal_no
capacity
availability

Entity: **Lounge**
Attributes:
lounge_id
type
terminal_no
capacity
availability

Note: type refers to the type of lounge (regular or premium).

Entity: **Terminal**

Attributes:

terminal_id
terminal_no

Entity: **Transportation**
Attributes:

vehicle_id
vehicle_type
capacity
status
source_terminal_no
destination_terminal_no

Note:- vehicle_type refers to the type of vehicle that could be either a normal bus or special vehicle for passengers with medical condition.

Relationship: **Passenger** accesses **Flight**
Cardinality: Many to Many
Participation:

Passenger has partial participation
Flight has partial participation

Relationship: **Employee** updates **Flight**
Cardinality: Many to Many
Participation:

Employee has partial participation
Flight has total participation

Note: Both the Ramp Planner and the ATC can update the flight details of the same flight.

Relationship: **Employee** logs into **Account**
Cardinality: One to One
Participation:

Employee has total participation
Account has total participation

Relationship: **Employee** updates **Parking**
Cardinality: Many to Many
Participation:
Passenger has partial participation
Parking has partial participation

Note: Here Employee refers to Facilities Manager

Relationship: **Employee** updates **Shops_Restaurants**
Cardinality: Many to Many
Participation:
Employee has partial participation
Shops_Restaurants has partial participation

Note: Here Employee refers to Facilities Manager

Relationship: **Employee** updates **Lounge**
Cardinality: Many to Many
Participation:
Passenger has partial participation
Lounge has partial participation

Note: Here Employee refers to Facilities Manager

Relationship: **Employee** updates **Transportation**
Cardinality: Many to Many
Participation:
Employee has partial participation
Transportation has partial participation

Note: Only Ground staff employees will be updating the transportation details.

LOGICAL DESIGN WITH NORMAL FORM IDENTIFICATION

Table: **Flight**
Columns:
flight_id
airline_no
airline_name

source
destination

Highest normalization level: 4NF

Indexes:

Index #: PRIMARY(flight_id) - clustered, idx_airline_no - Non Clustered

Columns: flight_id, airline_no

Justification:

flight_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy. Also, all the joins on Flight table will be done on flight_id.

airline_no - it is a good index to have as their will be frequent queries in the airport database based on airline_no.

create fulltext index idx_airline_no on Flight(airline_no);

Note :- We had Flight Table in Sprint 1 that had all the attributes. While normalisation, we separated it into two tables : Flight & FlightDetails.

Table : **FlightDetails**

Columns:

flight_id [foreign key:references Flight table]
date
departure_time
arrival_time
gate_no
terminal_no

Justification for PK : **flight_id+date** can uniquely identify any row of this table. If flight_id was primary key then we can not have multiple entries for same flight even if that flight is scheduled on daily basis.

Note: FlightDetails was separated from Flight so that ATC/Ramp planner can update departure_time and arrival_time and reduce data redundancy.

Highest normalization level: 4NF

Indexes:

Index #: PRIMARY (flight_id,date) - CLustered Index

Columns: flight_id,date

Justification:

flight_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy. It will also act as foreign key index as this is the first column to be sorted in the table and will make joins more efficient.

date is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy.

Table: **Passenger**

Columns:

passenger_id
first_name
last_name
passenger_type
email
phone_no

Justification: passenger_type refers to adult or child

Highest normalization level: 4NF

Indexes:

Index #: PRIMARY (passenger_id) - clustered
Columns: passenger_id
Justification:

passenger_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy. Also, all the joins on Passenger table will be done on passenger_id.

Justification: Passenger table is separated into two tables in process of normalization resulting tables are Passenger and PassengerFlight.

Table : **PassengerFlight**

Columns:

flight_id [foreign key references flight_id of Flight table]
passenger_id [foreign key references passenger_id of Passenger table]
pnr_no
ticket_no
class_id (foreign key references class_id of TicketClass table)

~~ticket_class~~ (created a new table to remove data redundancy)
seat_no
meal_type

Primary key : pnr_no+flight_id+passenger_id
Assuming that the PNR number for connecting flights might be same , we chose combination of these keys as a PK.

Highest normalization level:4NF
Indexes:
Index #: PRIMARY(pnr_no+flight_id+passenger_id) - clustered; passenger_id - non clustered; class_id - non clustered
Columns: pnr_no+flight_id+passenger_id; passenger_id; class_id
Justification:
pnr_no+flight_id+passenger_id is the composite primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this composite key makes search easy.
Non-Clustered Index:
Since class_id is the foreign key so creating indexes for this reduces the search process.
Since passenger_id is the foreign key so creating indexes for this reduces the search process.

Table : **TicketClass**
Columns:
class_id
ticket_class

Justification: ticket_class refers to Economy/Business/First-Class
Highest normalization level:4NF
Indexes:
Index #: PRIMARY(class_id) - clustered;
Columns: class_id
Justification:
class_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy. Also, all the joins on TicketClass table will be done on class_id.

Table: **Employee**
Columns:

employee_id
designation
emp_firstname
emp_lastname
street_address
city
state
country
zip_code
email
phone
salary

Note: Employee is a generalized table for all employees of the airport which includes an attribute designation which is basically a specialization of different types of employees. For example - Ramp Planner, ATC and Airport Administrator Support (Help Desk), etc.

Highest normalization level: 2NF
Justification : In Employee table columns street_address, city, state, country are dependent on zip_code , however considering the requirement of this specific database keeping all columns in one table is more efficient.

Indexes:
Index #: PRIMARY(employee_id) - clustered; idx_designation - non clustered
Columns: employee_id;designation
Justification:

employee_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy. Also, all the joins on Employee table will be done on employee_id.

Designation - We have one generalised table for all the employees and most of the queires are based on designation of the employee.

create fulltext index idx_designation on Employee(designation);

Table: **Account**
Columns:
account_id

username
password
emp_id [foreign key; references employee_id of Employee Table]

Highest normalization level: 4NF
Indexes:
Index #: PRIMARY(account_id);emp_id (employee_id)
Columns: account_id;emp_id
Justification:
account_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy.
Non-Clustered Index:
Since emp_id is the foreign key so creating indexes for this reduces the search process.

Table: **ShopCategory**
Columns:
category_id
name

Highest normalization level: 4NF
Justification : ShopCategory was added to remove the dependency of category and to reduce data redundancy in ShopRestCat table.
Indexes:
Index #: PRIMARY (category_id)- clustered
Columns: category_id
Justification: category_id is the PK and all joins from ShopRestCat table will be performed using category_id.

Table: **ShopRestCat**
Columns:
shop_id
name
category_id (foreign key references category_id of ShopCategory table)

Highest normalization level: 4NF
Justification : ShopRestCat was added to remove the dependency of name and category and to reduce data redundancy

Indexes:

Index #: PRIMARY(shop_id) - clustered; category_id - non clustered
Columns: shop_id, category_id
Justification:

shop_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy and joins will be efficient as well.

Category_id is the foreign key and having an index on this attribute will make joins more efficient.

Table: **ShopsRestaurants**

Columns:

shop_id foreign key references ShopRestCat(shop_id)
terminal_no
area

Highest normalization level: 4NF
Justification : Table ShopsRestaurants was separated into two tables in order to remove the dependency resulting table ShopsRestaurants is in 4nf.
Indexes:

Index #: PRIMARY(shop_id, terminal_no) - clustered
Columns: shop_id, terminal
Justification:

terminal_no is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy.

Since shop_id is the foreign key & primary key so creating indexes for this reduces the search process and makes join also efficient.

Table: **LoungeCat**

Columns:

lounge_id
type

Highest normalization level: 4NF
Justification : LoungeCat was separated from Lounge to reduce data redundancy and address the dependency of lounge_id and lounge type

Indexes:

Index #: PRIMARY (lounge_id) - clustered
Columns: lounge_id

Justification:

lounge_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy and joins will be efficient.

Table:**Lounge**

Columns:

lounge_terminal_id
lounge_id (foreign key, references lounge_id of LoungeCat)
terminal_no
capacity
availability

Primary Key : lounge_terminal_id+lounge_id+terminal_no

Combination of these 3 keys has been chosen as primary key to put a constraint that only entry of a lounge type is allowed in a terminal.

Highest normalization level: 4NF
Justification : Table Lounge was separated into two tables in order to eliminate dependency, resulting table Lounge in 4nf.

Indexes:

Index#:PRIMARY(lounge_terminal_id+lounge_id+terminal_no) - clustered ; lounge_terminal_id - non clustered

Columns: lounge_terminal_id, lounge_id, terminal_no
Justification:

lounge_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy and joins become more efficient.

terminal_no is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy.

Non-Clustered Index:

Since lounge_terminal_id is the foreign key & primary so creating two indexes for this reduces the search process and and joins become more efficient. (One clustered; one non clustered)

Table: **Parking**

Columns:

lot_id
terminal_no
capacity
availability

Highest normalization level: 4NF

Note: We had many to many relationships for this sprint, which translates to each of the below separate tables in our Relational Model.

Indexes:

Index #: PRIMARY(lot_id) - clustered
Columns: lot_id
Justification:

lot_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy and jons become more efficient.

Table: **EmpFlight**
Columns:

flight_id [foreign key references lot_id of Flight table]
employee_id [foreign key references employee_id of Employee table]
last_modified_at
row_version

Highest normalization level: 4NF
Indexes:

Index #: PRIMARY (flight_id, employee_id) - clustered;
Employee_id - non clustered
Columns: flight_id, employee_id
Justification:

Flight_id+employee_id is the composite primary key of the table. Having an index on this makes the search operations efficient.
Both of them are foreign keys as well. However, the first column is already sorted based on the primary key. So we have an index for the second column i.e. employee_id that will make joins very efficient.

Table: **EmpParking**
Columns:

lot_id [foreign key references lot_id of Parking table]
employee_id [foreign key references employee_id of Employee table]
last_modified_at
row_version

Highest normalization level: 4NF
Indexes:
Index #: PRIMARY (lot_id, employee_id) - clustered;
Employee_id - non clustered
Columns: lot_id, employee_id
Justification:

lot_id+employee_id is the composite primary key of the table. Having an index on this makes the search operations efficient.
Both of them are foreign keys as well. However, the first column is already sorted based on the primary key. So we have an index for the second column i.e. employee_id that will make joins very efficient.

Table: **EmpShopRest**
Columns:
shop_id [foreign key references shop_id of ShopRestCat table]
employee_id [foreign key references employee_id of Employee table]
last_modified_at
row_version

Highest normalization level: 4NF
Indexes:
Index #: PRIMARY (shop_id, employee_id) - clustered;
Employee_id - non clustered
Columns: shop_id, employee_id
Justification:

shop_id+employee_id is the composite primary key of the table. Having an index on this makes the search operations efficient.
Both of them are foreign keys as well. However, the first column is already sorted based on the primary key. So we have an index for the second column i.e. employee_id that will make joins very efficient.

Table: **EmpLounge**
Columns:
lounge_terminal_id [foreign key references lounge_terminal_id of Lounge table]
employee_id [foreign key references employee_id of Employee table]

last_modified_at
row_version

Highest normalization level: 4NF
Indexes:
Index #: PRIMARY (lounge_terminal_id, employee_id) - clustered; Employee_id - non clustered
Columns: lounge_terminal_id, employee_id
Justification:

lounge_termnal_id+employee_id is the composite primary key of the table. Having an index on this makes the search operations efficient.
Both of them are foreign keys as well. However, the first column is already sorted based on the primary key. So we have an index for the second column i.e. employee_id that will make joins very efficient.

Note : We decided not to have a table for Terminal as it doesn't have any defining attribute other than terminal number. So, we chose to have terminal_no as an attribute wherever required in our database.

Table : **VehicleType**
Columns :
vehicle_type_id
vehicle_type

Note - vehicle_type refers to (Normal/Emergency/Medical)

Highest normalization level: 4NF
Indexes:
Index #: PRIMARY (vehicle_type_id)- clustered
Columns: vehicle_type_id
Justification:

Vehicle_type_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy and joins become more efficient.

Table: **Transportation**

Columns:

vehicle_id

vehicle_type_id (foreign key references vehicle_type_id of VehicleType table)

capacity

availability

source_terminal

destination_terminal

Highest normalization level: 4NF

Indexes:

Index #: PRIMARY (vehicle_id) - clustered, vehicle_type_id - non clustered

Columns: vehicle_id, vehicle_type_id

Justification:

Vehicle_id is the primary key in this table and data in the table are ordered in the same way as the primary key. So, creating an index for this column makes search easy and joins become more efficient.

Vehicle_type_id is the foreign key, having an index on this will make joins faster.

Table: **EmpTrans**

Columns:

vehicle_id[foreign key references vehicle_id of Vehicle table]

employee_id [foreign key references employee_id of Employee table]

last_modified_at

row_version

Highest normalization level: 4NF

Indexes:

Index #: PRIMARY (vehicle_id, employee_id) - clustered;
vehicle_id- non clustered

Columns: vehicle_id, employee_id

Justification:

vehicle_id+employee_id is the composite primary key of the table. Having an index on this makes the search operations efficient.

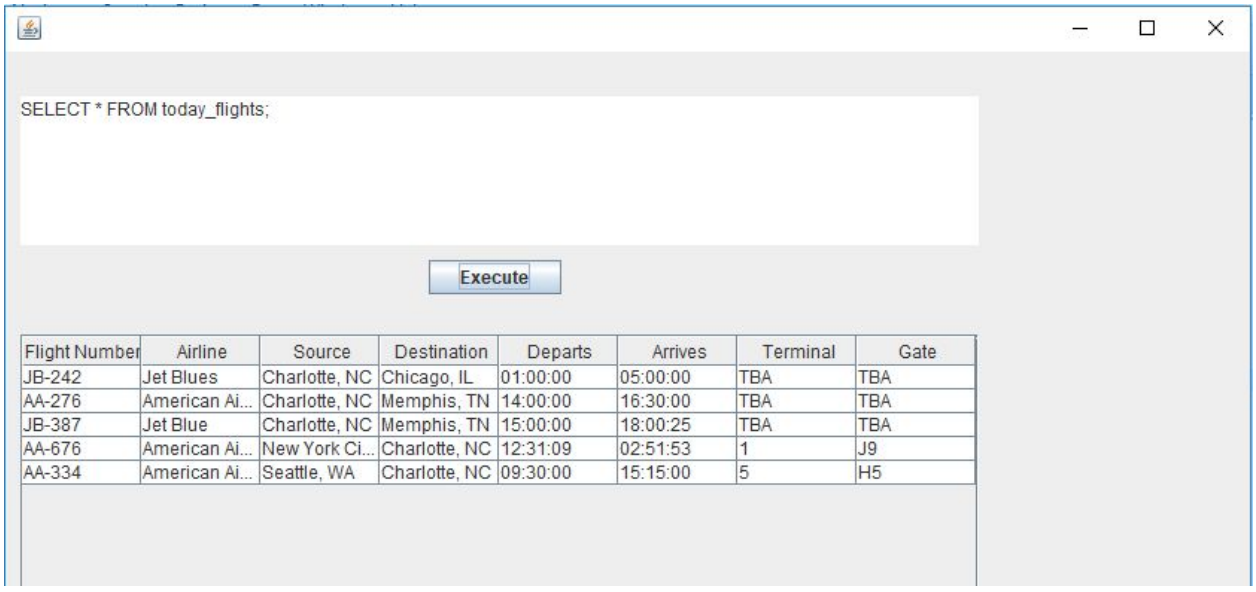
Both of them are foreign keys as well. However, the first column is already sorted based on the primary key. So we have an index for the second column i.e. vehicle_id in this case, that will make joins very efficient.

VIEWS AND STORED PROGRAMS

View: today_flights
Goal: This view is for the Airport Administrator Support (Help Desk). In this view AAS employees can see all the relevant details of current date flight and inform the passengers.
Create Statement:

```
create view today_flights as
select f.airline_no as 'Flight Number', f.airline_name as Airline ,f.source as Source,
f.destination as Destination, fd.departure_time as 'Departs',fd.arrival_time 'Arrives',
ifnull(fd.terminal_no, 'TBA') as 'Terminal', ifnull(fd.gate_no, 'TBA') as Gate
from Flight f
inner join FlightDetails fd using(flight_id)
where fd.date = curdate()
order by source, departure_time;
```

Output:

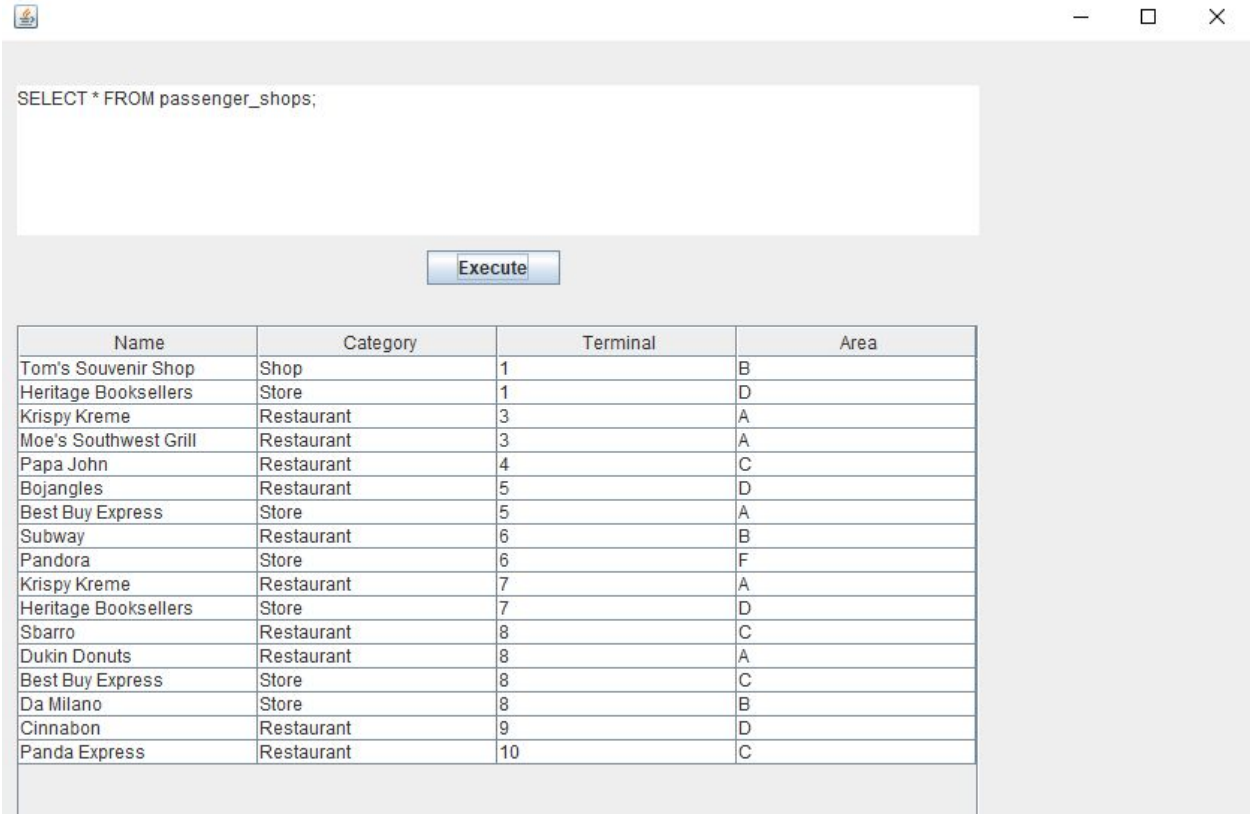


Flight Number	Airline	Source	Destination	Departs	Arrives	Terminal	Gate
JB-242	Jet Blues	Charlotte, NC	Chicago, IL	01:00:00	05:00:00	TBA	TBA
AA-276	American Ai...	Charlotte, NC	Memphis, TN	14:00:00	16:30:00	TBA	TBA
JB-387	Jet Blue	Charlotte, NC	Memphis, TN	15:00:00	18:00:25	TBA	TBA
AA-676	American Ai...	New York Ci...	Charlotte, NC	12:31:09	02:51:53	1	J9
AA-334	American Ai...	Seattle, WA	Charlotte, NC	09:30:00	15:15:00	5	H5

View: passenger_shops
Goal: This view is for the passengers availing facilities of the airport. It will provide the details of shops or restaurants for the airport along with the terminal and area details so that they can easily navigate where ever they want.
Create Statement:

```
create view passenger_shops as
select s.name as 'Name' ,sc.name 'Category',sr.terminal_no 'Terminal', sr.area 'Area'
from ShopRestCat s
inner join ShopsRestaurants sr using(shop_id)
inner join ShopCategory sc using(category_id)
order by terminal_no;
```

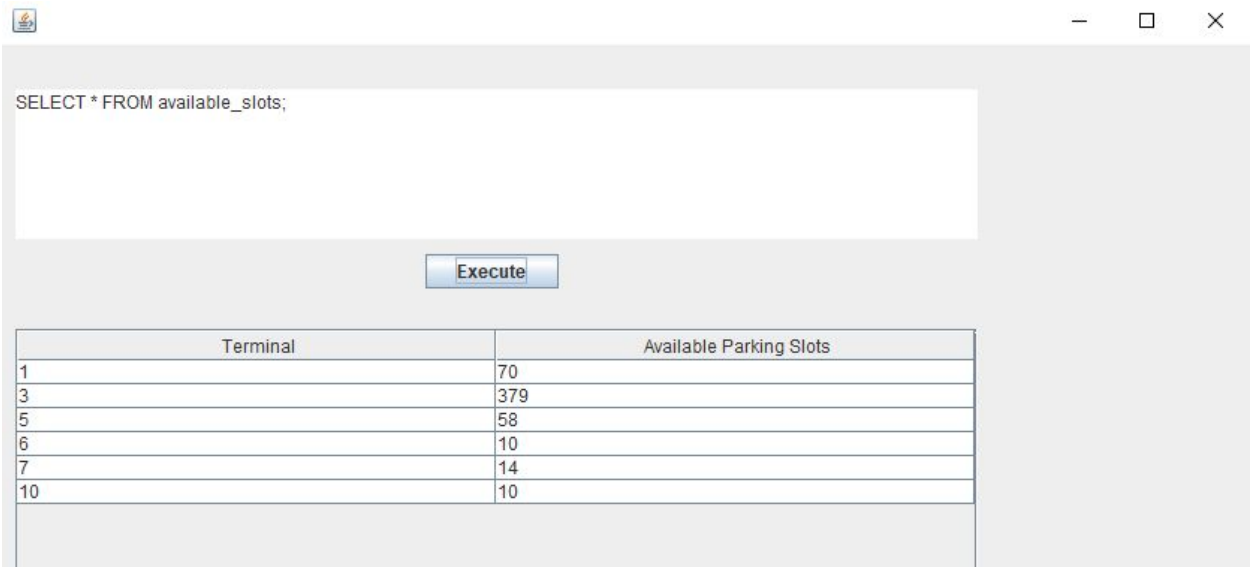
Output:



Name	Category	Terminal	Area
Tom's Souvenir Shop	Shop	1	B
Heritage Booksellers	Store	1	D
Krispy Kreme	Restaurant	3	A
Moe's Southwest Grill	Restaurant	3	A
Papa John	Restaurant	4	C
Bojangles	Restaurant	5	D
Best Buy Express	Store	5	A
Subway	Restaurant	6	B
Pandora	Store	6	F
Krispy Kreme	Restaurant	7	A
Heritage Booksellers	Store	7	D
Sbarro	Restaurant	8	C
Dukin Donuts	Restaurant	8	A
Best Buy Express	Store	8	C
Da Milano	Store	8	B
Cinnabon	Restaurant	9	D
Panda Express	Restaurant	10	C

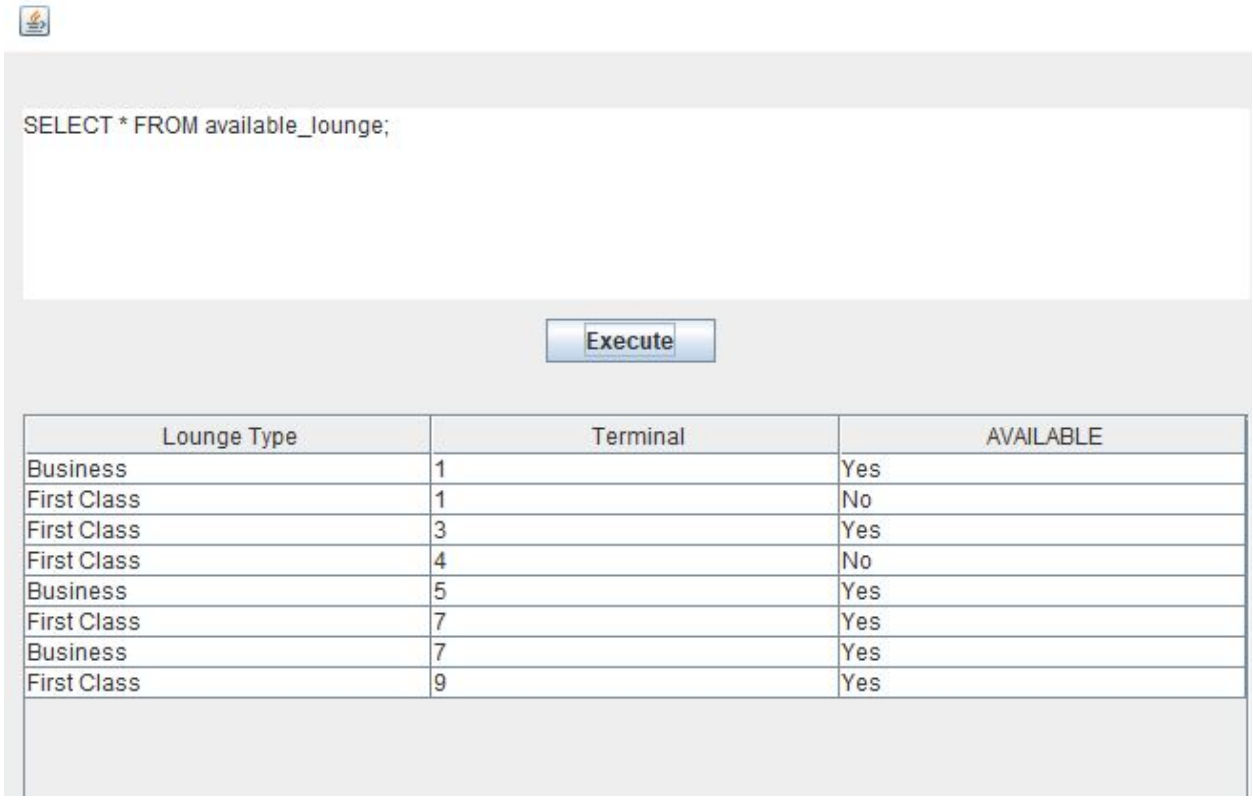
View: available_slots
Goal: This view is for the Airport Facilities Managers, which will give them

the overview availability of the available parking slots in a terminal (summing up availability of all the lots)
Create Statement:
create view available_slots as
select terminal_no as 'Terminal', sum(availability) as 'Available Parking Slots' from Parking
group by terminal_no;
Output:



View: available_lounge
Goal: This view is for the passengers travelling by Business or First-Class, which provides them details about the Lounges in different terminals of the airport with their availability using the stored function 'Availability'
Create Statement:
create view available_lounge as
select lc.type as 'Lounge Type', l.terminal_no as 'Terminal',
Availability(l.availability) as 'AVAILABLE' from Lounge l
inner join LoungeCat lc using(lounge_id)
order by terminal_no;

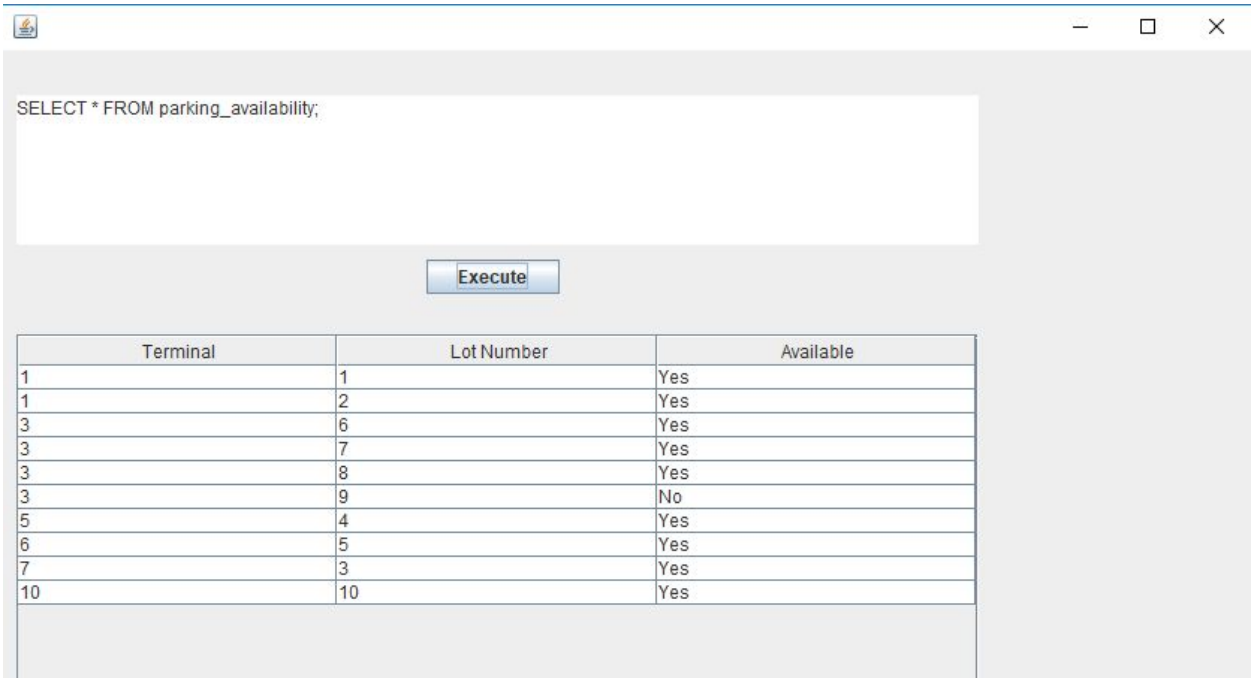
Output:



Lounge Type	Terminal	AVAILABLE
Business	1	Yes
First Class	1	No
First Class	3	Yes
First Class	4	No
Business	5	Yes
First Class	7	Yes
Business	7	Yes
First Class	9	Yes

View: parking_availability
Goal: This view is for the passengers, which will provide them details about the available parking slots with in the terminals. We have used stored function 'Availability' for this view as well.
Create Statement:
create view parking_availability as
select terminal_no as 'Terminal', lot_id as 'Lot Number',Availability(availability) as 'Available'
from Parking order by terminal_no;

Output:



Events

Event: One-time event
Name: update_empflight_event
Goal: We added new columns last_modified_at and row_version in this sprint for EmpFlight table. We created this one-time event to update the last_modified_at and row_version to 1 for this table. These attributes will be updated in future by the triggers created as part of this sprint.
Creation Statement:

```
CREATE EVENT update_empflight_event
ON SCHEDULE AT CURRENT_TIMESTAMP
ON COMPLETION PRESERVE
DO
  UPDATE EmpFlight set last_modified_at =NOW() , row_version=1;
```

```
1  SELECT * FROM EmpFlight;
2
```

Result Grid | Filter Rows: | Edit:

flight_id	employee_id	last_modified_at	row_version
1	11	2018-11-24 14:46:50	1
2	20	2018-11-24 14:46:50	1
3	11	2018-11-24 14:46:50	1
5	9	2018-11-24 14:46:50	1
7	9	2018-11-24 14:46:50	1
9	11	2018-11-24 14:46:50	1
10	20	2018-11-24 14:46:50	1
12	9	2018-11-24 14:46:50	1
15	11	2018-11-24 14:46:50	1
15	20	2018-11-24 14:46:50	1
16	9	2018-11-24 14:46:50	1
21	20	2018-11-24 14:46:50	1
NULL	NULL	NULL	NULL

SELECT * FROM EmpFlight ;

Execute

flight_id	employee_id	last_modified_at	row_version
1	11	2018-11-24 14:46:50.0	1
2	20	2018-11-24 14:46:50.0	1
3	11	2018-11-24 14:46:50.0	1
5	9	2018-11-24 14:46:50.0	1
7	9	2018-11-24 14:46:50.0	1
9	11	2018-11-24 14:46:50.0	1
10	20	2018-11-24 14:46:50.0	1
12	9	2018-11-24 14:46:50.0	1
13	9	2018-11-26 06:15:12.0	3
13	10	2018-11-26 06:12:03.0	2
14	9	2018-11-26 05:52:29.0	1
15	11	2018-11-24 14:46:50.0	1
15	20	2018-11-24 14:46:50.0	1
16	9	2018-11-24 14:46:50.0	1
21	20	2018-11-24 14:46:50.0	1

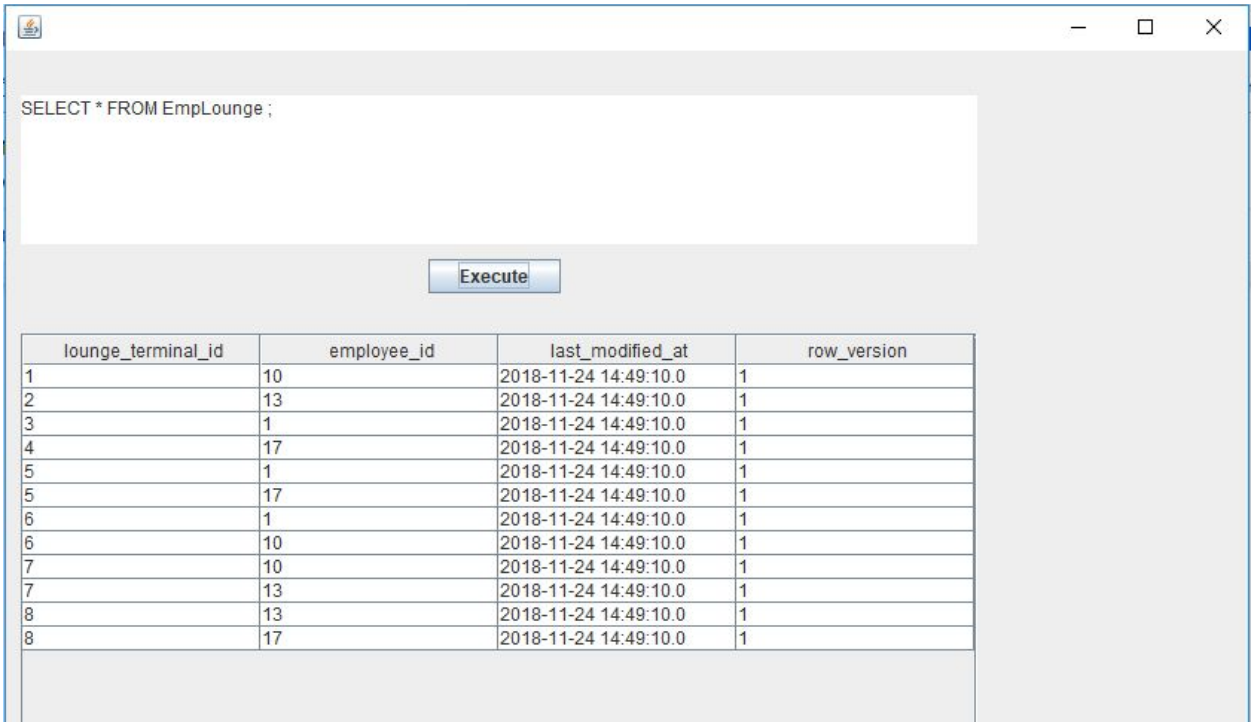
Event: One-time event
Name: update_emplounge_event
Goal: We added new columns last_modified_at and row_version in this sprint for EmpLounge table. We created this one-time event to update the last_modified_at and row_version to 1 for this table. These attributes will be updated in future by the triggers created as part of this sprint.
Creation Statement:

```
CREATE EVENT update_emplounge_event
ON SCHEDULE AT CURRENT_TIMESTAMP
ON COMPLETION PRESERVE
DO
  UPDATE EmpLounge set last_modified_at =NOW() , row_version=1;
```

```
1 • SELECT * FROM ADBMS.EmpLounge;
```

Result Grid | Filter Rows: | Edit:

lounge_terminal_id	employee_id	last_modified_at	row_version
1	10	2018-11-24 14:49:10	1
2	13	2018-11-24 14:49:10	1
3	1	2018-11-24 14:49:10	1
4	17	2018-11-24 14:49:10	1
5	1	2018-11-24 14:49:10	1
5	17	2018-11-24 14:49:10	1
6	1	2018-11-24 14:49:10	1
6	10	2018-11-24 14:49:10	1
7	10	2018-11-24 14:49:10	1
7	13	2018-11-24 14:49:10	1
8	13	2018-11-24 14:49:10	1
8	17	2018-11-24 14:49:10	1
NULL	NULL	NULL	NULL



lounge_terminal_id	employee_id	last_modified_at	row_version
1	10	2018-11-24 14:49:10.0	1
2	13	2018-11-24 14:49:10.0	1
3	1	2018-11-24 14:49:10.0	1
4	17	2018-11-24 14:49:10.0	1
5	1	2018-11-24 14:49:10.0	1
5	17	2018-11-24 14:49:10.0	1
6	1	2018-11-24 14:49:10.0	1
6	10	2018-11-24 14:49:10.0	1
7	10	2018-11-24 14:49:10.0	1
7	13	2018-11-24 14:49:10.0	1
8	13	2018-11-24 14:49:10.0	1
8	17	2018-11-24 14:49:10.0	1

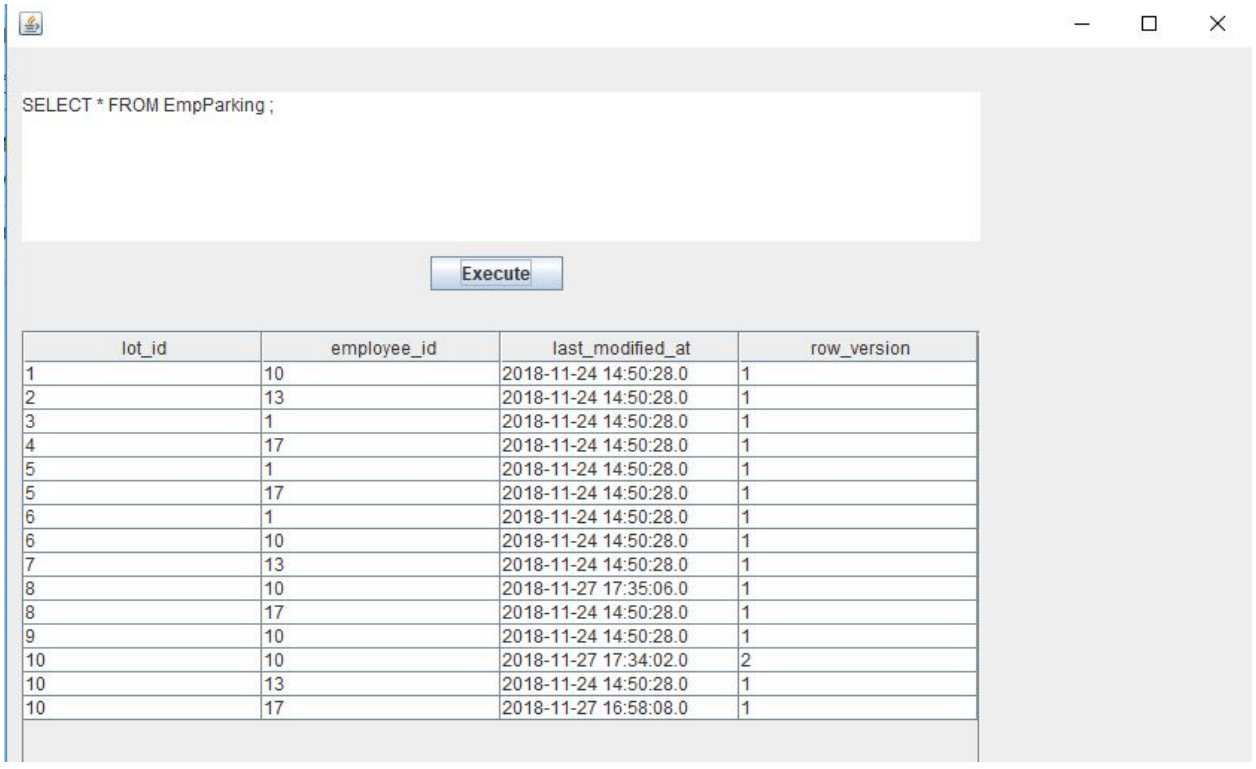
Event: One-time event
Name: update_empparking_event
Goal: We added new columns last_modified_at and row_version in this sprint for EmpParking table. We created this one-time event to update the last_modified_at and row_version to 1 for this table. These attributes will be updated in future by the triggers created as part of this sprint.
Creation Statement:

```
CREATE EVENT update_empparking_event
ON SCHEDULE AT CURRENT_TIMESTAMP
ON COMPLETION PRESERVE
DO
  UPDATE EmpParking set last_modified_at =NOW() , row_version=1;
```

```
14
15 •  select * from EmpParking;
```

Result Grid | Filter Rows: | Edit:

lot_id	employee_id	last_modified_at	row_version
1	10	2018-11-24 14:50:28	1
2	13	2018-11-24 14:50:28	1
3	1	2018-11-24 14:50:28	1
4	17	2018-11-24 14:50:28	1
5	1	2018-11-24 14:50:28	1
5	17	2018-11-24 14:50:28	1
6	1	2018-11-24 14:50:28	1
6	10	2018-11-24 14:50:28	1
7	13	2018-11-24 14:50:28	1
8	17	2018-11-24 14:50:28	1
9	10	2018-11-24 14:50:28	1
10	13	2018-11-24 14:50:28	1
NULL	NULL	NULL	NULL

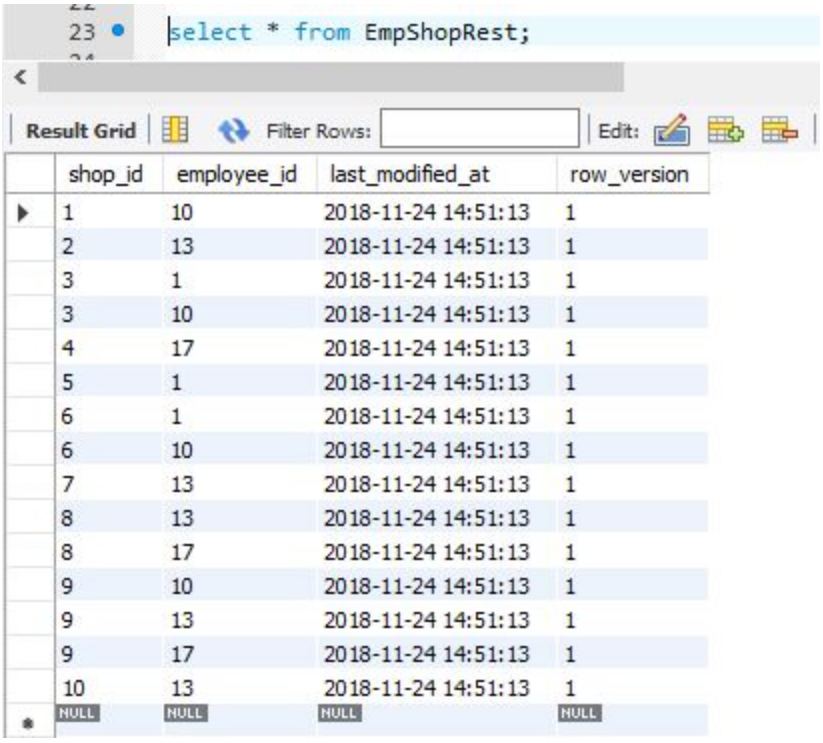


lot_id	employee_id	last_modified_at	row_version
1	10	2018-11-24 14:50:28.0	1
2	13	2018-11-24 14:50:28.0	1
3	1	2018-11-24 14:50:28.0	1
4	17	2018-11-24 14:50:28.0	1
5	1	2018-11-24 14:50:28.0	1
5	17	2018-11-24 14:50:28.0	1
6	1	2018-11-24 14:50:28.0	1
6	10	2018-11-24 14:50:28.0	1
7	13	2018-11-24 14:50:28.0	1
8	10	2018-11-27 17:35:06.0	1
8	17	2018-11-24 14:50:28.0	1
9	10	2018-11-24 14:50:28.0	1
10	10	2018-11-27 17:34:02.0	2
10	13	2018-11-24 14:50:28.0	1
10	17	2018-11-27 16:58:08.0	1

Event: One-time event
Name: update_empshoprest_event
Goal: We added new columns last_modified_at and row_version in this sprint for EmpShopRest table. We created this one-time event to update the last_modified_at

and row_version to 1 for this table. These attributes will be updated in future by the triggers created as part of this sprint.
Creation Statement:

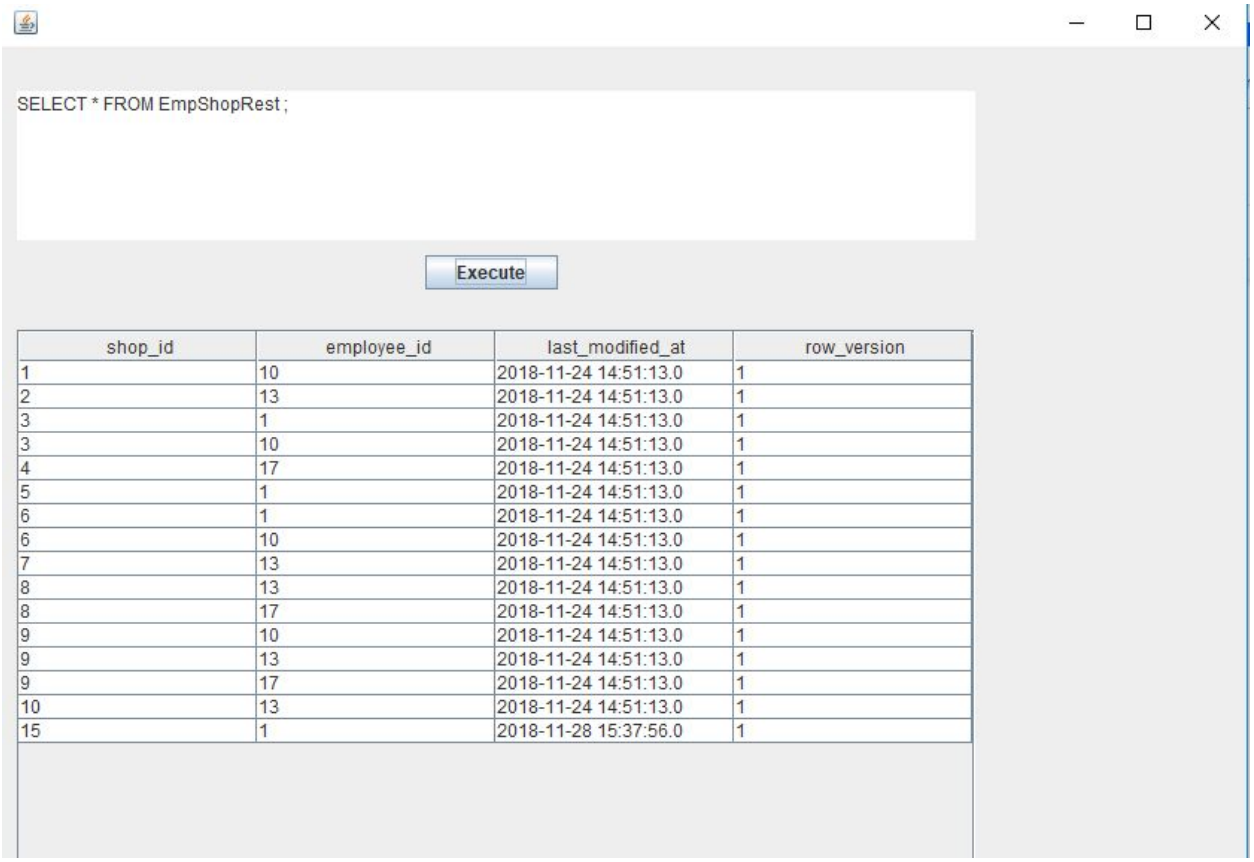
```
CREATE EVENT update_empshoprest_event
ON SCHEDULE AT CURRENT_TIMESTAMP
ON COMPLETION PRESERVE
DO
   UPDATE EmpShopRest set last_modified_at =NOW() , row_version=1;
```



```
23 • select * from EmpShopRest;
```

Result Grid | Filter Rows: | Edit:

shop_id	employee_id	last_modified_at	row_version
1	10	2018-11-24 14:51:13	1
2	13	2018-11-24 14:51:13	1
3	1	2018-11-24 14:51:13	1
3	10	2018-11-24 14:51:13	1
4	17	2018-11-24 14:51:13	1
5	1	2018-11-24 14:51:13	1
6	1	2018-11-24 14:51:13	1
6	10	2018-11-24 14:51:13	1
7	13	2018-11-24 14:51:13	1
8	13	2018-11-24 14:51:13	1
8	17	2018-11-24 14:51:13	1
9	10	2018-11-24 14:51:13	1
9	13	2018-11-24 14:51:13	1
9	17	2018-11-24 14:51:13	1
10	13	2018-11-24 14:51:13	1
NULL	NULL	NULL	NULL



shop_id	employee_id	last_modified_at	row_version
1	10	2018-11-24 14:51:13.0	1
2	13	2018-11-24 14:51:13.0	1
3	1	2018-11-24 14:51:13.0	1
3	10	2018-11-24 14:51:13.0	1
4	17	2018-11-24 14:51:13.0	1
5	1	2018-11-24 14:51:13.0	1
6	1	2018-11-24 14:51:13.0	1
6	10	2018-11-24 14:51:13.0	1
7	13	2018-11-24 14:51:13.0	1
8	13	2018-11-24 14:51:13.0	1
8	17	2018-11-24 14:51:13.0	1
9	10	2018-11-24 14:51:13.0	1
9	13	2018-11-24 14:51:13.0	1
9	17	2018-11-24 14:51:13.0	1
10	13	2018-11-24 14:51:13.0	1
15	1	2018-11-28 15:37:56.0	1

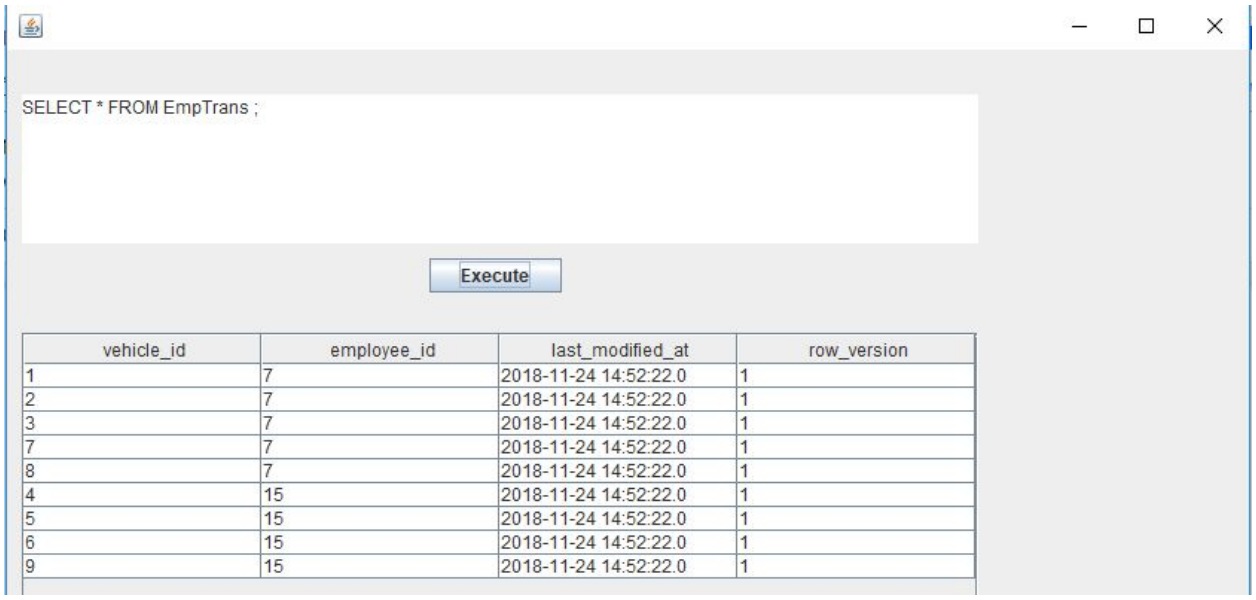
Event: One-time event
Name: update_emptrans_event
Goal: We added new columns last_modified_at and row_version in this sprint for EmpTrans table. We created this one time event to update the last_modified_at and row_version to 1 for this table. These attributes will be updated in future by the triggers created as part of this sprint.
Creation Statement:

```
CREATE EVENT update_emptrans_event
ON SCHEDULE AT CURRENT_TIMESTAMP
ON COMPLETION PRESERVE
DO
  UPDATE EmpTrans set last_modified_at =NOW() , row_version=1;
```

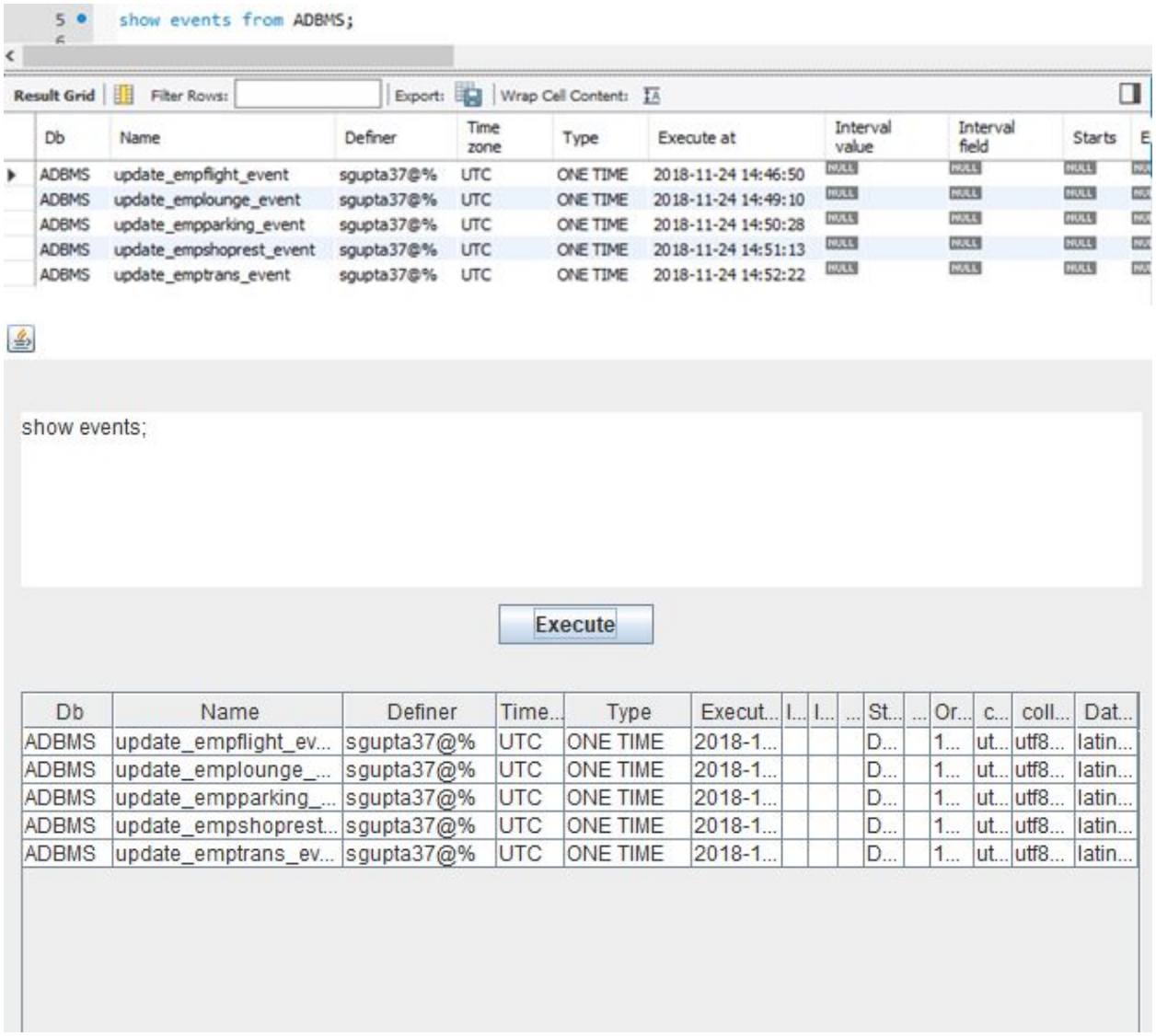
```
30
31 •  select * from EmpTrans;
```

Result Grid | Filter Rows: | Edit:

vehicle_id	employee_id	last_modified_at	row_version
1	7	2018-11-24 14:52:22	1
2	7	2018-11-24 14:52:22	1
3	7	2018-11-24 14:52:22	1
7	7	2018-11-24 14:52:22	1
8	7	2018-11-24 14:52:22	1
4	15	2018-11-24 14:52:22	1
5	15	2018-11-24 14:52:22	1
6	15	2018-11-24 14:52:22	1
9	15	2018-11-24 14:52:22	1
NULL	NULL	NULL	NULL



Note : Since we created all the events with 'ON COMPLETION PRESEVE' clause, here is the event history :



Stored Function:

Stored function: Availability
Parameters: Input is an integer i.e. availability and output return varchar availability as 'YES' or 'No'
Goal: We have created this stored function to determine availability. If the availability is greater than 0, then Yes is returned else No. This stored function will be used by views where we will be displaying this information to the users.

```
DELIMITER $$
CREATE FUNCTION Availability(a int) RETURNS VARCHAR(10)
   DETERMINISTIC
BEGIN
```

```
   DECLARE available varchar(10);

   IF a > 0 THEN
              SET available = 'YES';
   ELSE
              SET available = 'NO';
        END IF;
 RETURN (available);
END $$
DELIMITER ;
```

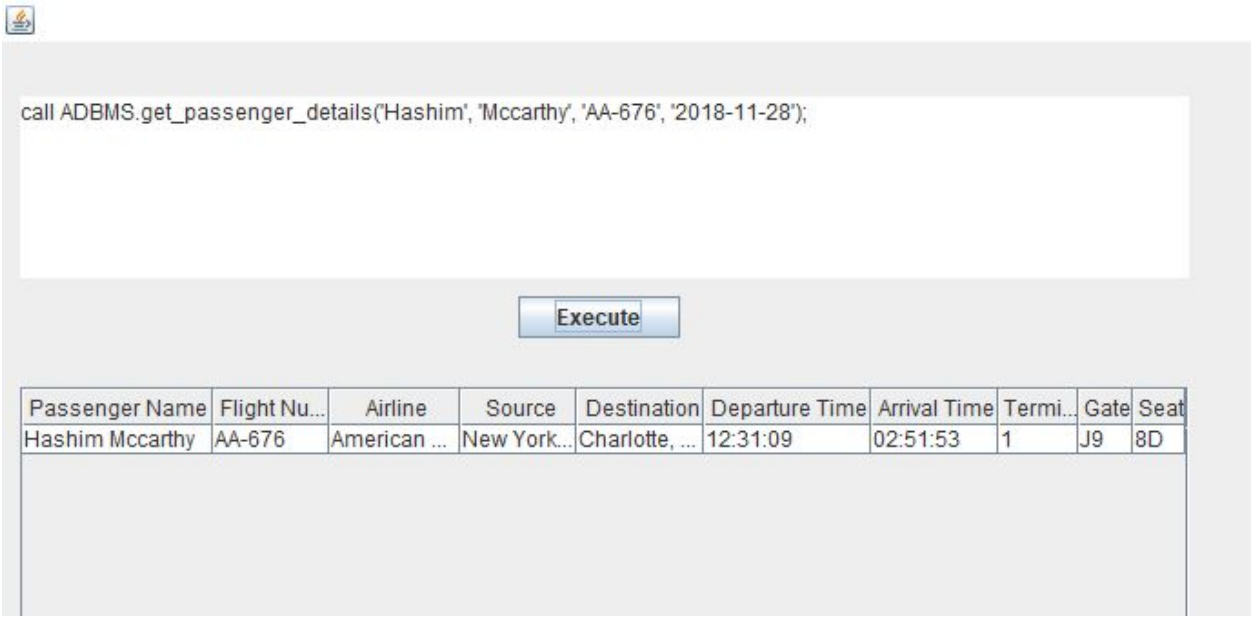
Stored Procedures:

Stored procedure: get_passenger_details
Parameters: IN – passenger first name, IN – passenger last name, IN – airline flight no., IN – flight date
Goal: This is procedure is for the passengers to query flight details. They just need to give their first name, last name ,flight no & flight date and they will get all the details about their upcoming flight.

Create Statement:

```
DELIMITER //
CREATE PROCEDURE get_passenger_details(IN f_name varchar(255), l_name varchar(255), airline varchar(255))
 BEGIN
 select concat_ws(' ', p.first_name,p.last_name) as 'Passenger Name', f.airline_no as 'Flight Number',
f.airline_name as 'Airline', f.source as 'Source', f.destination as 'Destination',
fd. departure_time 'Departure Time' , fd.arrival_time as 'Arrival Time',
ifnull(fd.terminal_no, 'TBA') as 'Terminal', ifnull(fd.gate_no, 'TBA') as 'Gate', pf.seat_no as 'Seat'
from Passenger p
inner join PassengerFlight pf using(passenger_id)
inner join FlightDetails fd using(flight_id)
inner join Flight f using(flight_id)
where p.first_name = f_name and p.last_name = l_name and f.airline_no = airline and fd.date = curdate() ;
 END //
DELIMITER ;
```

Output:



Stored procedure: login
Parameters: IN – username, IN – password, OUT - login status (successfull or unsuccessful)
Goal: This is procedure is for airport employees to login to the system. It takes their username and password and validates with the account table. If the username & password are correct, then Status is a successful message else, it fails.

Create Statement:

```
CREATE PROCEDURE `login`(IN user varchar(255), passwd varchar(20), out Status varchar(255))
 BEGIN
  case
   when exists(select * from Account where username = user and password = passwd)
  then set Status = 'Login Successful';
  else set Status= 'Invalid Username or Password !!';
  end case;
   END
```

Output :

```
1  set @Status = '0';
2  call ADBMS.login('OCEAN123', 'DSDS2323', @Status);
3  select @Status;
4
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

@Status
Login Successful

```
3  call login('OCEAN123','DSDS233',@Status);
4  select @status;
```

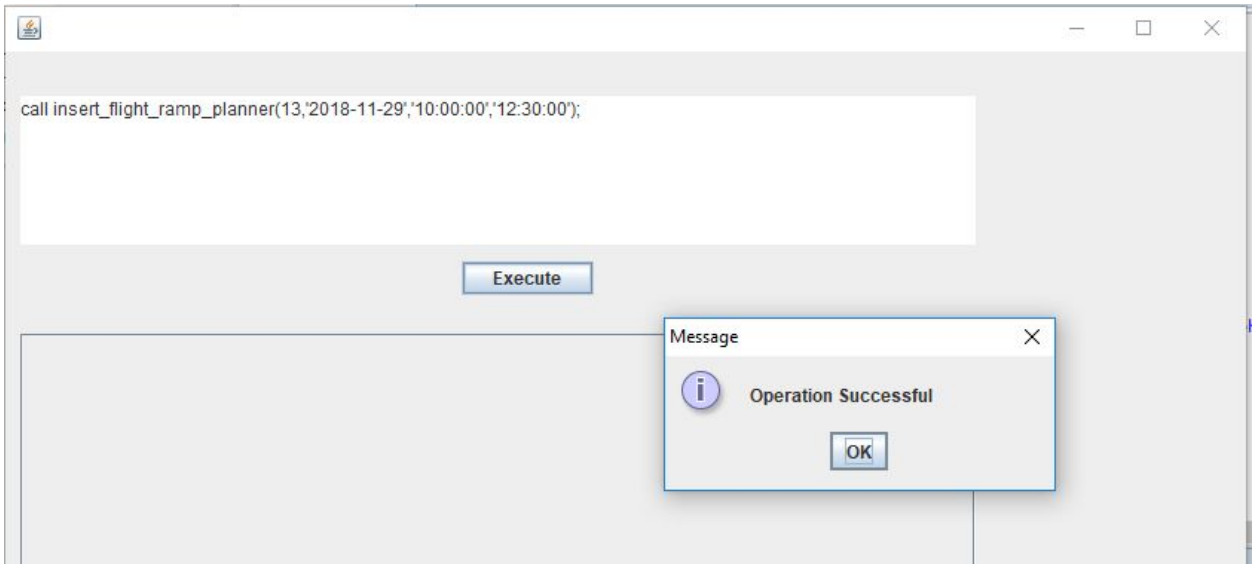
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

@status
Invalid Username or Password !!

Stored procedure: insert_flight_ramp_planner
Parameters: IN – flight id, IN – flight date, IN – departure time, IN arrival time
Goal: This procedure is for the ramp planners to insert the flight details in the airport database. They can insert the flight departure time, arrival time & date. However, they are not allowed to update the terminal & gate information.
Create Statement:

```
DELIMITER //
CREATE PROCEDURE insert_flight_ramp_planner(IN id int, flight_date date, d_time time, a_time time)
 BEGIN
 Insert into FlightDetails(flight_id,date,depature_time,arrival_time) values
 (id,flight_date,d_time,a_time);
  END //
DELIMITER ;
```

Output:
call insert_flight_ramp_planner(13,'2018-11-29','10:00:00','12:30:00');

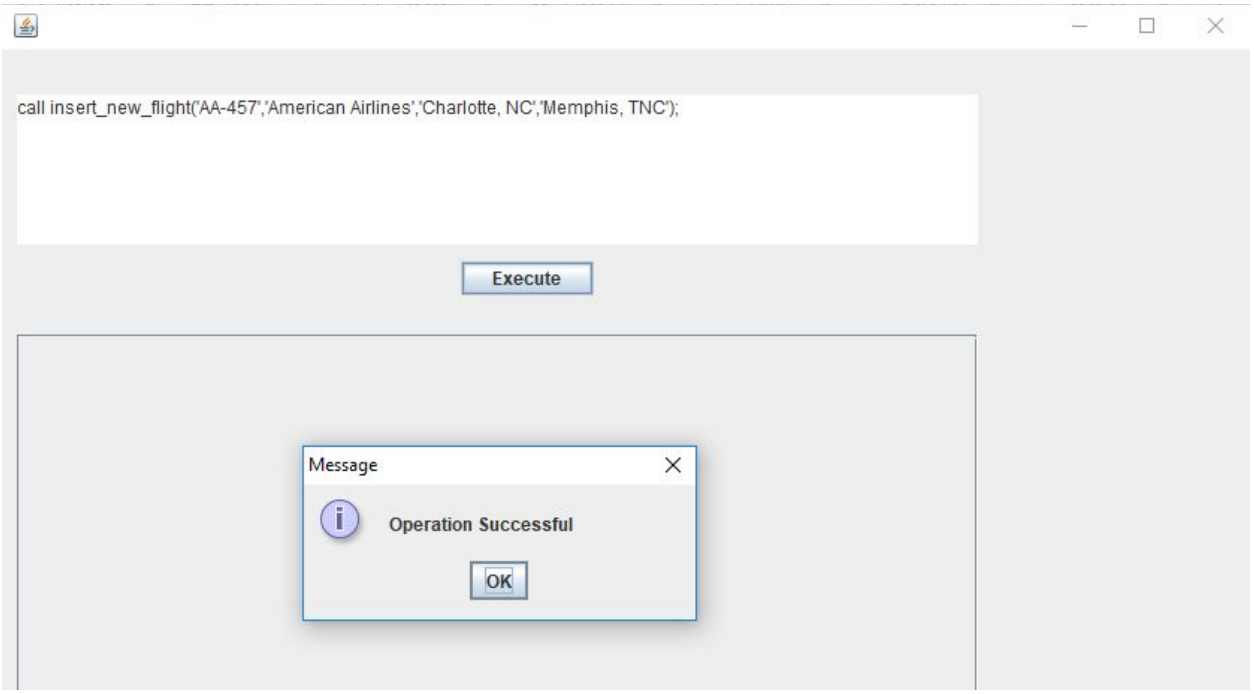


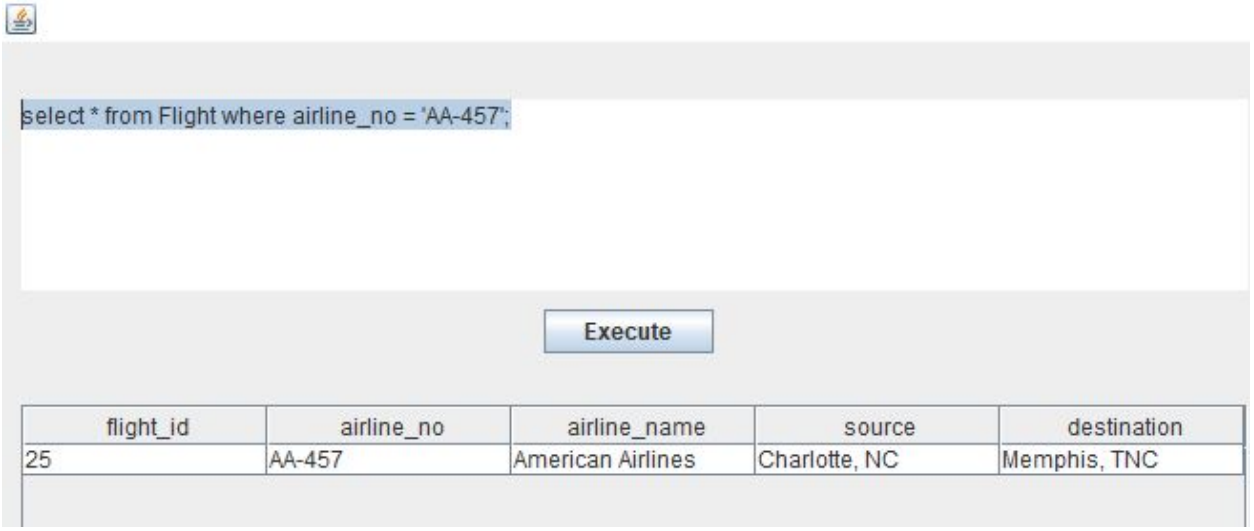
Stored procedure: insert_new_flight
Parameters: IN – airline no., IN – airline name, IN – source, IN destination
Goal: This procedure is for the ramp planners to update the newly introduced to airport database. This will update the Flight table with all the basic information of the flight given in IN parameters.
Create Statement:

```
DELIMITER //
CREATE PROCEDURE insert_new_flight(IN airline_number varchar(255), name varchar(255), source_city varchar(255), destination_city varchar(255) )
 BEGIN
 Insert into Flight(airline_no,airline_name,source,destination) values
 (airline_number,name, source_city, destination_city);
  END //
DELIMITER ;
```

Output:

```
call insert_new_flight('AA-457','American Airlines','Charlotte, NC','Memphis, TNC');
```



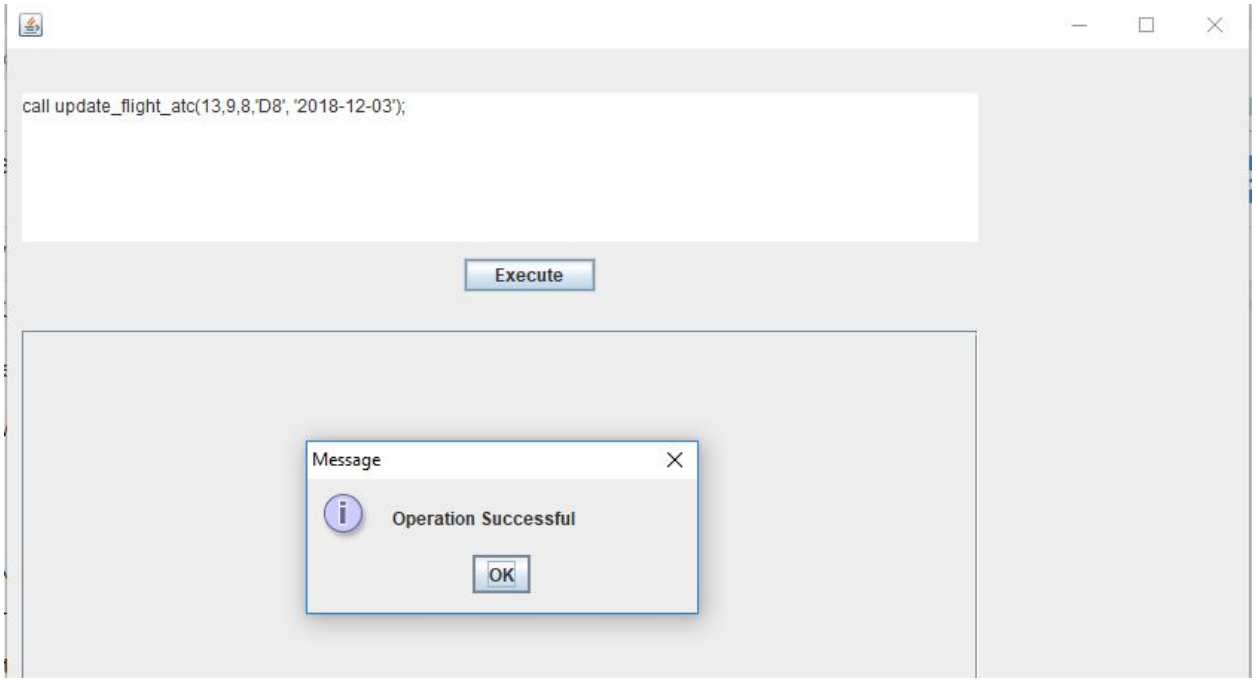


Stored procedure: update_flight_atc
Parameters: IN – flight id, IN – employee id, IN – terminal no., IN – gate no., IN – flight date
Goal: This procedure is for the ATC Employees of the airport to update the gate & terminal for the departing or arriving flights for the input date. This procedure will also update the EmpFlight table with the details of which employee updated this information. If it is a new entry, then it will be inserted, else no action. (Trigger will take care of updating last_modified_at & row_version)
Create Statement:

```
CREATE PROCEDURE update_flight_atc(IN id int, emp int, terminal int, gate varchar(10), flight_date date)
 BEGIN
 Update FlightDetails set terminal_no=terminal, gate_no=gate where flight_id = id and date = flight_date;
 CASE
        WHEN NOT EXISTS (
                SELECT flight_id, employee_id
        FROM EmpFlight
                WHERE `flight_id` = `id`
        AND `employee_id` =`emp`
        )
        THEN
                INSERT INTO EmpFlight
                                (flight_id, employee_id)
                                values (id,emp);
                        ELSE
                                BEGIN
DECLARE row int;
                                Select row_version into @row from EmpFlight where flight_id=id and employee_id=emp;
                                Update EmpFlight set last_modified_at = now(), row_version=@row+1 where flight_id=id and employee_id=emp ;
                                END;

        END CASE;

  END //
DELIMITER ;
```

Output:

```
call update_flight_atc(13,9,8,'D8', '2018-12-03');
```



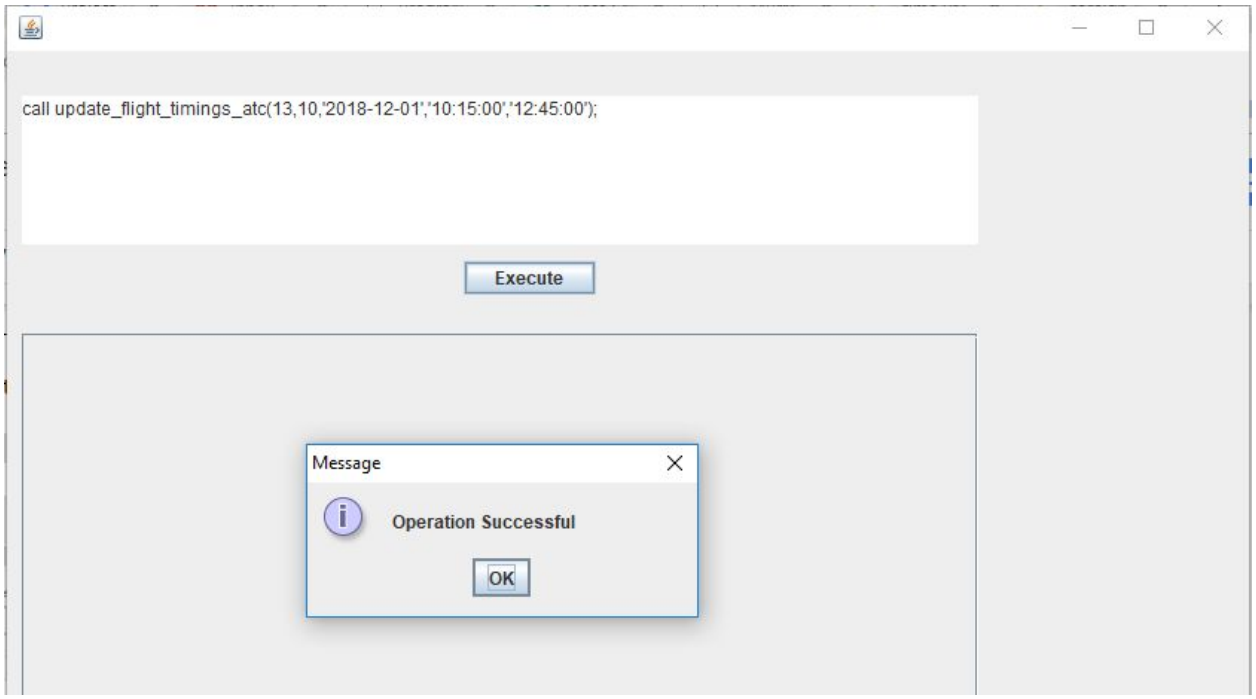
Stored procedure: update_flight_timings_atc
Parameters: IN – flight id, IN – employee id, IN – flight date, IN – departure time, IN – arrival time
Goal: This procedure is for the ATC Employees to update any change in arriving or departing flight timings. Again, EmpFlight table will be updated like previous procedure.
Create Statement:

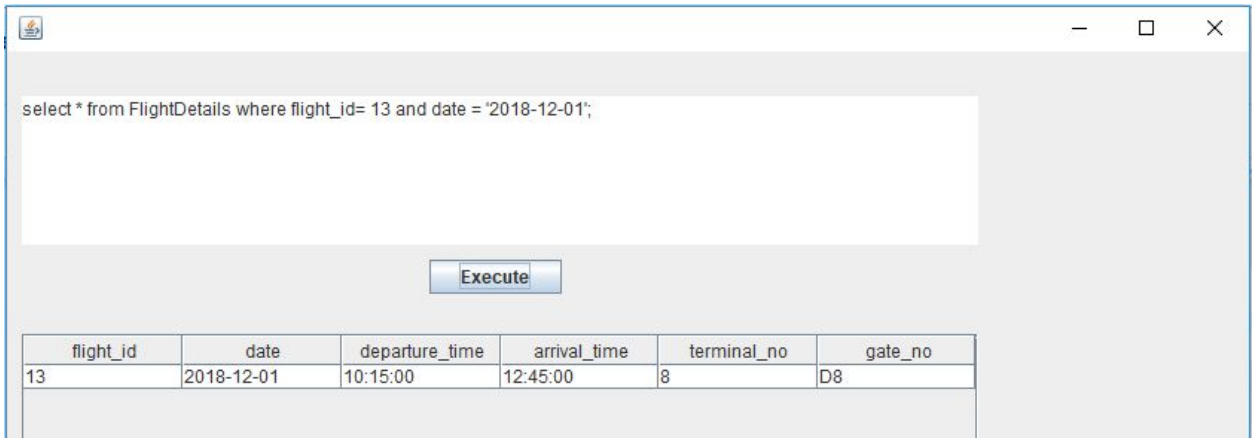
```
DELIMITER //
CREATE PROCEDURE update_flight_timings_atc(IN id int, emp int,flight_date date, d_time time, a_time time)
 BEGIN
  Update FlightDetails set departure_time=d_time, arrival_time=a_time
  where flight_id = id and date = flight_date;
 CASE
        WHEN NOT EXISTS (
                SELECT flight_id, employee_id
        FROM EmpFlight
                WHERE `flight_id` = `id`
        AND `employee_id` =`emp`
```

```
        )
        THEN
                INSERT INTO EmpFlight
                        (flight_id, employee_id)
                        values (id,emp);
                    ELSE
                        BEGIN
DECLARE row int;
                        Select row_version into @row from EmpFlight where flight_id=id and employee_id=emp;
                        Update EmpFlight set last_modified_at = now(), row_version=@row+1 where flight_id=id and employee_id=emp ;

                        END;

        END CASE;

  END //
DELIMITER ;
```

Output:

call update_flight_timings_atc(13,10,'2018-12-01','10:15:00','12:45:00');





Stored procedure: add_shop
Parameters: IN – employee id, IN – shop name, IN – category IN – terminal, IN - area
Goal: This procedure is for the airport facilities manager to add new shop to the database.
Create statement :

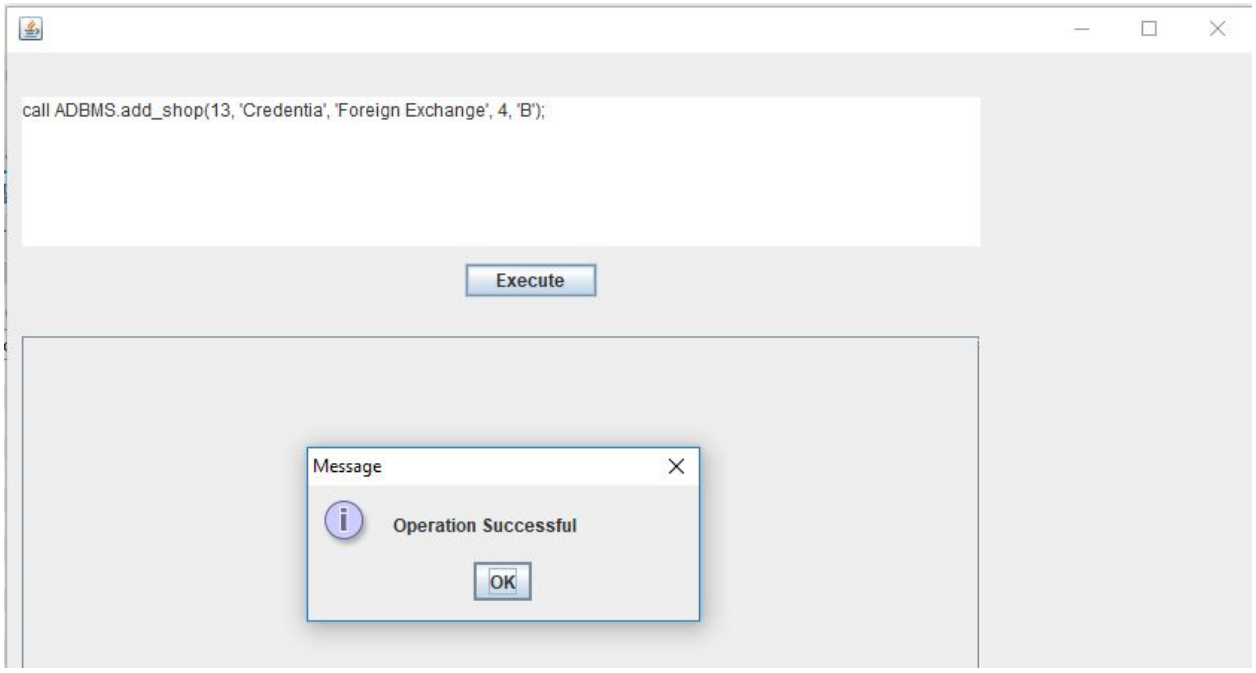
Create Statement :

```
DELIMITER //
create procedure add_shop(
in emp int(11),
in shop_name varchar(225),
in cat varchar(225),
in terminal int(11),
in area varchar(225)
```

```
)
begin
Declare id int;
Declare category int;
case when exists(select category_id from ShopCategory where name=cat)
then
        select category_id into @category from ShopCategory where name=cat;
else
        begin
        insert into ShopCategory(name) values(cat);
        select category_id into @category from ShopCategory where name = cat;
   end;
end case;

insert into ShopRestCat(name,category_id) values (shop_name,@category);
select shop_id into @id from ShopRestCat where name=shop_name;
insert into ShopsRestaurants(shop_id,terminal_no,area) values
(@id,terminal,area);
CASE
        WHEN NOT EXISTS (
                SELECT shop_id, employee_id
       FROM EmpShopRest
                WHERE shop_id = @id
       AND employee_id =emp
          )
          THEN
             INSERT INTO EmpShopRest
                                (shop_id, employee_id)
                                values (@id,emp);
                        ELSE
                                BEGIN
             DECLARE row int;
               Select row_version into @row from EmpShopRest where shop_id=@id
and employee_id=emp;
                                 Update EmpShopRest set last_modified_at = now(),
row_version=@row+1
             where shop_id=@id and employee_id=emp ;
                                END;

          END CASE;
end //
DELIMITER ;
```

Output :

call ADBMS.add_shop(13, 'Credentia', 'Foreign Exchange', 4, 'B');



Store Procedure : update_shop
Parameters : IN - shop id, IN - employee id, IN - shop name, IN - shop category
Goal - This procedure is to update shop information in the ShopRestCat table and update the EmpShopRest table with employee information making the changes.

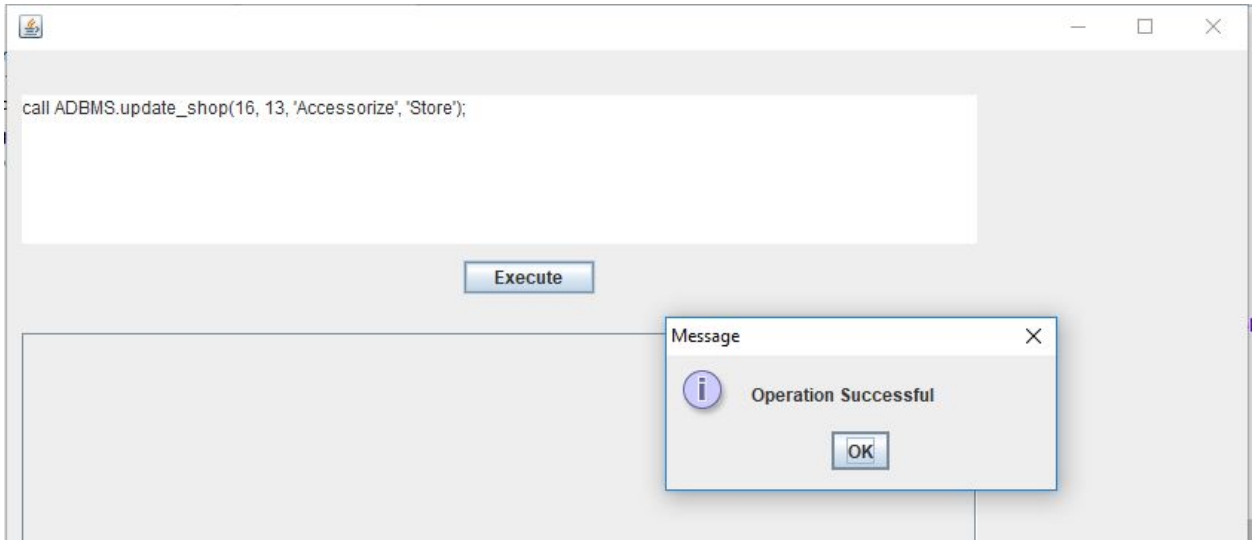
Create Statement :

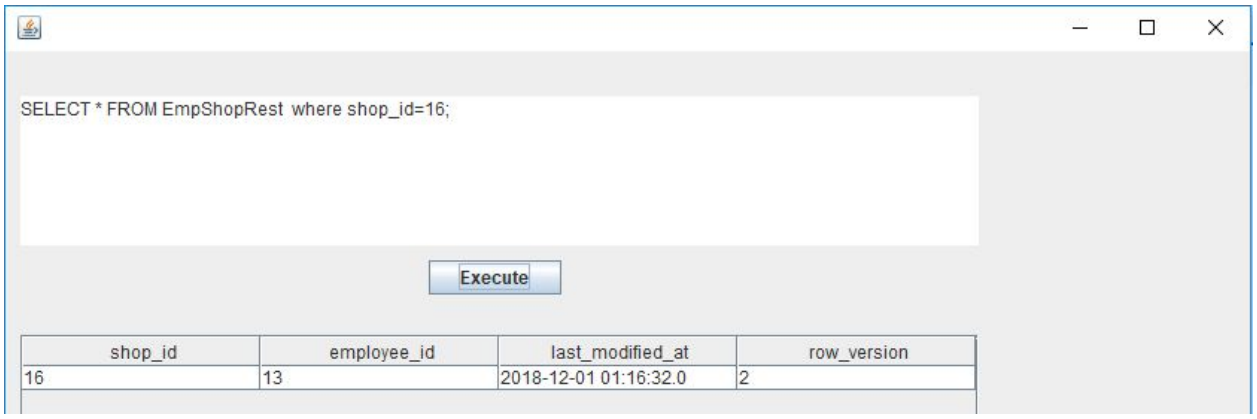
```
DELIMITER //
create procedure update_shop(
In id int(11),
in emp int,
in shop_name varchar(225),
in cat varchar(225)
)
begin
update ShopRestCat
set name = shop_name, category_id  = (select category_id from ShopCategory where name=cat)
where shop_id = id;
CASE
        WHEN NOT EXISTS (
                SELECT shop_id, employee_id
        FROM EmpShopRest
                WHERE shop_id = id
        AND employee_id =emp
           )
           THEN
              INSERT INTO EmpShopRest
```

```
                        (shop_id, employee_id)
                        values (id,emp);
                ELSE
                        BEGIN
        DECLARE row int;
         Select row_version into @row from EmpShopRest where shop_id=id and
employee_id=emp;
                         Update EmpShopRest set last_modified_at = now(),
row_version=@row+1
        where shop_id=id and employee_id=emp ;
                        END;

    END CASE;
END //
DELIMITER ;
```

Output:

call ADBMS.update_shop(16, 13, 'Accessorize', 'Store');





Stored Procedure: update_lounge
Parameters : IN - lounge id, IN - employee id, IN - available
Goal - This procedure updates the lounge availability in the database and updates EmpLounge table with the details of employee making this change.

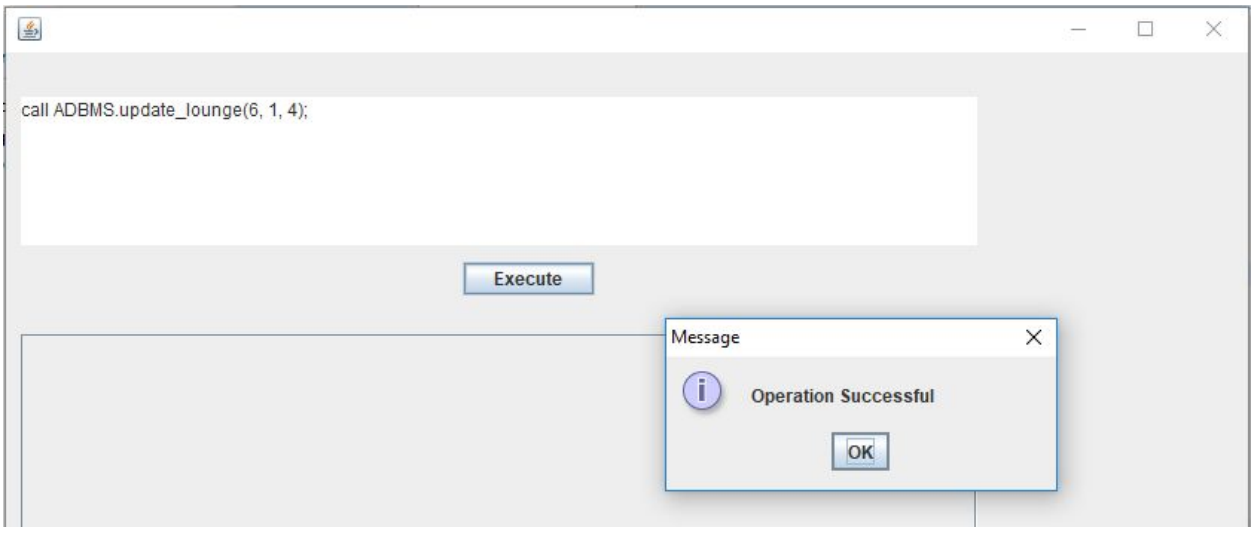
Create statement :

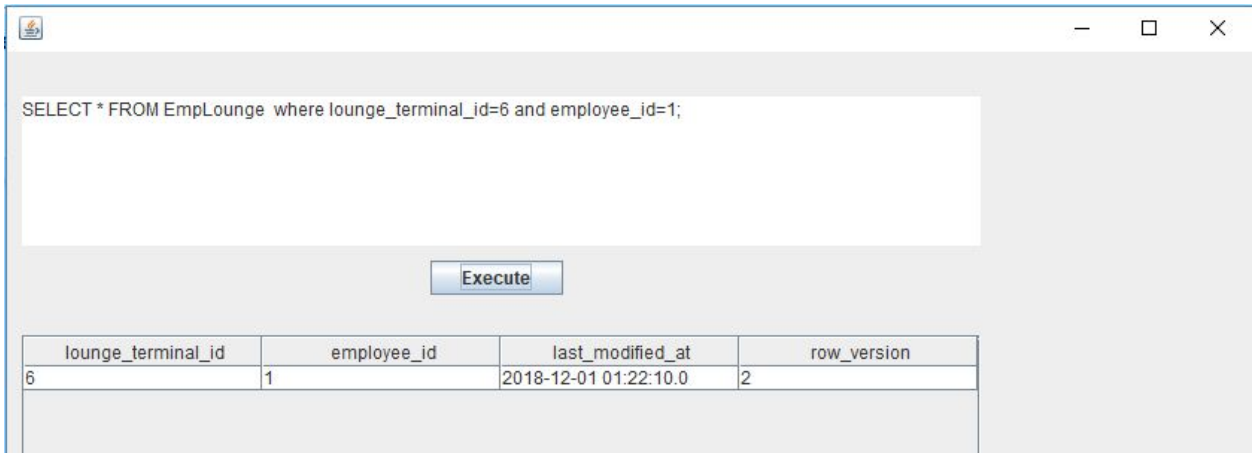
```
DELIMITER //
create procedure update_lounge(
in id int(11),
in emp int,
in var_availability int(11)
)
BEGIN
update Lounge
set availability = var_availability
where lounge_terminal_id = id;
CASE
        WHEN NOT EXISTS (
                SELECT lounge_terminal_id, employee_id
```

```
        FROM EmpLounge
                WHERE lounge_terminal_id = id
        AND employee_id =emp
           )
           THEN
              INSERT INTO EmpLounge
                            (lounge_terminal_id, employee_id)
                            values (id,emp);
                        ELSE
                            BEGIN
              DECLARE row int;
                           Select row_version into @row from EmpLounge where
lounge_terminal_id=id and employee_id=emp;
              Update EmpLounge set last_modified_at = now(), row_version=@row+1
              where lounge_terminal_id=id and employee_id=emp ;
                            END;

           END CASE;
END //
DELIMITER ;
```

Output :

```
call ADBMS.update_lounge(6, 1, 4);
```





Stored Procedure : update_parking
Parameters - IN - lot_id, IN - employee id, IN - available
Goal - This procedure is to update the parking availability for terminal and update EmpParking table with the details of employee making the update.

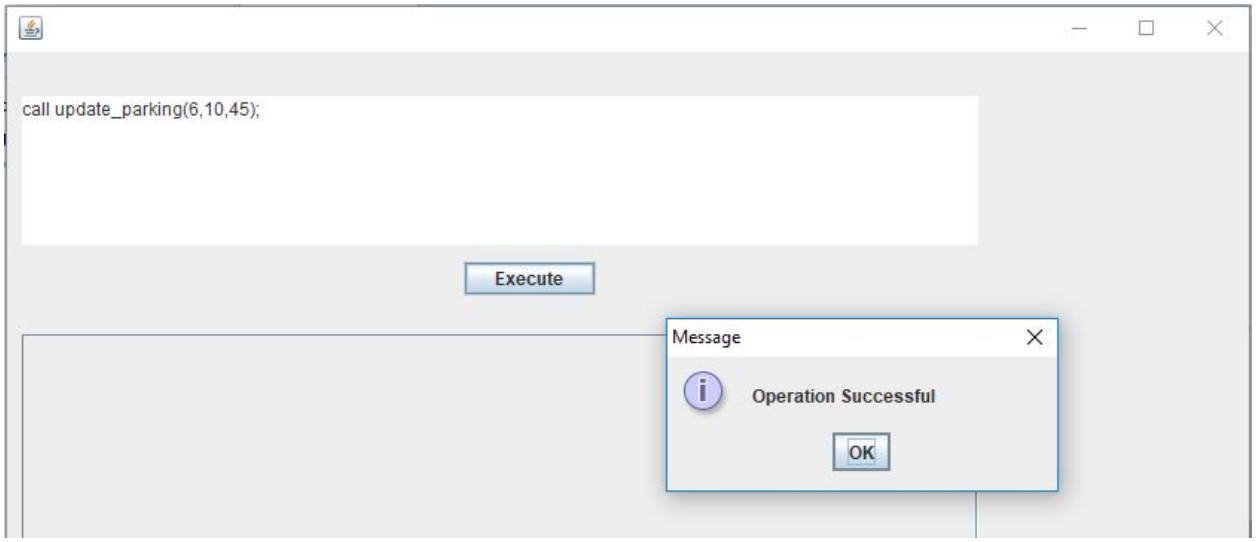
Create Statement :

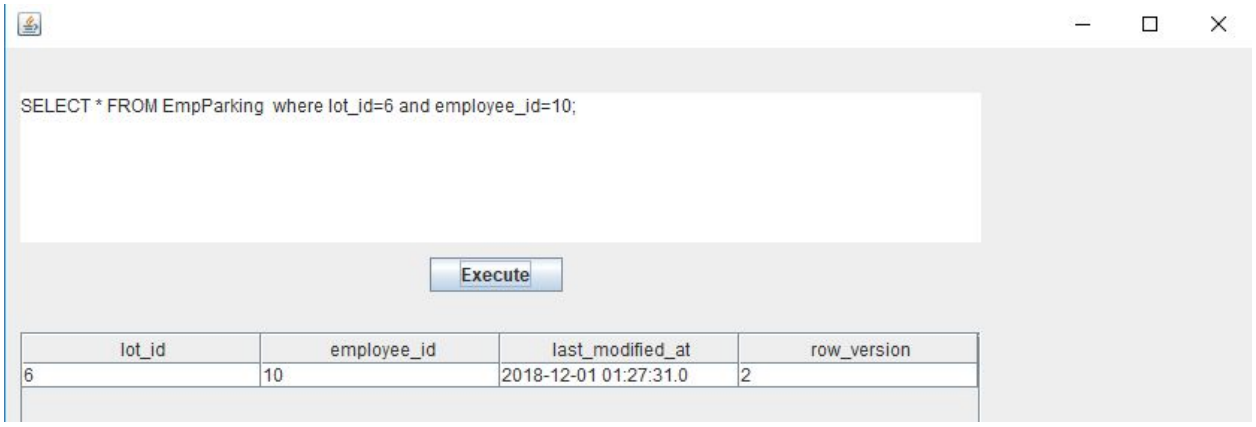
```
DELIMITER //
CREATE PROCEDURE update_parking(IN id int, emp int, available int)
 BEGIN
 Update Parking set availability=available where lot_id = id;
 CASE
        WHEN NOT EXISTS (
                SELECT *
       FROM EmpParking
                WHERE lot_id = id
       AND employee_id =emp
          )
          THEN
```

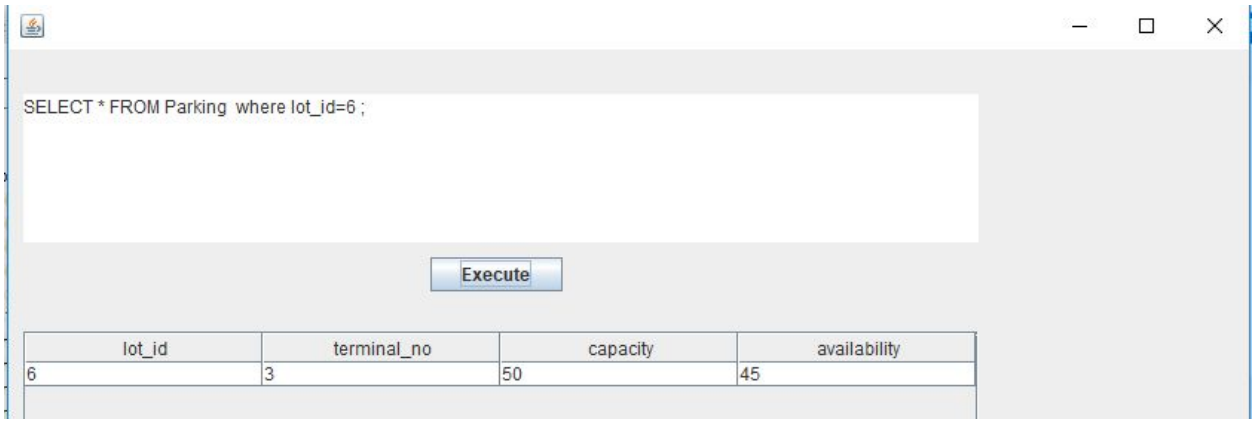
```
            INSERT INTO EmpParking
                        (lot_id, employee_id)
                        values (id,emp);
                  ELSE
                        BEGIN
            DECLARE row int;
                Select row_version into @row from EmpParking where lot_id=id and
employee_id=emp;
                              Update EmpParking set last_modified_at = now(),
row_version=@row+1
            where lot_id=id and employee_id=emp ;
                        END;

         END CASE;

  END //
DELIMITER ;
```

Output :

call update_parking(6,10,45);





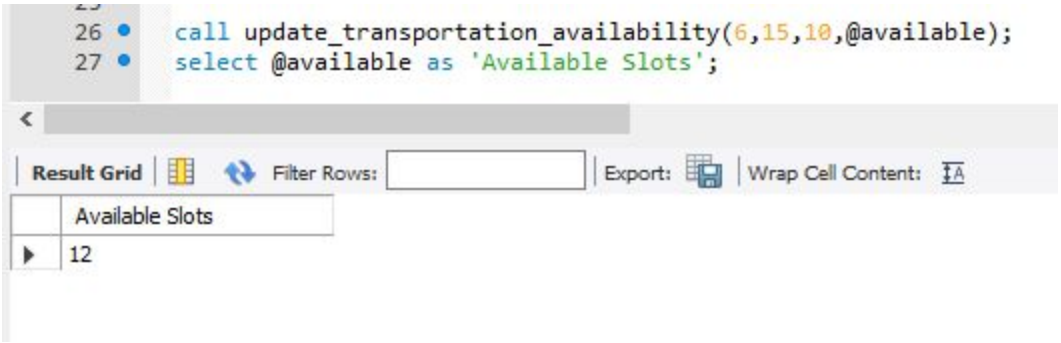


Stored procedure: update_transportation_availability
Parameters: IN – vehicle id, IN – employee id, IN – remaining available seats, OUT – updated available slots
Goal: This procedure is for the airport ground staff to update the remaining available seats for airport transportation vehicles and display the updated availability.This table will also update the EmpTrans table if there is no entry in the table for employee & vehicle, else trigger will update the last_modified_at & row_version
Create Statement:

```
DELIMITER //
CREATE PROCEDURE update_transportation_availability(IN id int, IN emp int, IN set_availability int, OUT available int)
BEGIN
Update Transportation set availability = set_availability
where vehicle_id = id;
select availability into @available from Transportation;
CASE
        WHEN NOT EXISTS (
```

```
            SELECT vehicle_id, employee_id
        FROM EmpTrans
            WHERE vehicle_id = id
        AND employee_id =emp
        )
        THEN
            INSERT INTO EmpTrans
                        (vehicle_id, employee_id)
                        values (id,emp);
                    ELSE
                        BEGIN
DECLARE row int;
            Select row_version into @row from EmpTrans where vehicle_id=id and
employee_id=emp;
            Update EmpTrans set last_modified_at = now(), row_version=@row+1
            where vehicle_id=id and employee_id=emp ;
                        END;

        END CASE;

  END //
DELIMITER ;
```

Output:



Triggers:

Trigger: insert_emp_flight on EmpFlight

Goal:

Trigger serves as the default method for updating last_modified_at column in EmpFlight whenever new tuples are added to the table EmpFlight using a datetime data type. The purpose of this trigger is to show when an employee makes a

change to a Flight. This trigger also sets the row_version column to 1 by default to show no changes have been made to the table.

Trigger: insert_emp_shop on EmpShopRest
Goal:
Trigger serves as the default method for updating last_modified_at & increment row version column in EmpShopRest whenever new tuples are added to the table ShopRestCat & ShopsRestaurant. The purpose of this trigger is to show when these tuples were added.

Trigger: insert_emp_lounge on EmpLounge
Goal:
Trigger serves as the default method for updating last_modified_at & increment row version column in EmpLounge whenever new tuples are added to the table Lounge. The purpose of this trigger is to show when these tuples were added.

Trigger: insert_emp_parking on EmpParking
Goal:
Trigger serves as the default method for updating last_modified_at column & increment row version column in EmpParking whenever new tuples are added to the table Parking. The purpose of this trigger is to show when these tuples were added.

Trigger: insert_emp_transp on EmpTrans
Goal:
Trigger serves as the default method for updating last_modified_at & increment row version column in EmpTrans whenever new tuples are added to the table Transportation. The purpose of this trigger is to show when these tuples were added.